Detecting Encrypted Command & Control Channels with Network Fingerprints

Luukas Larinkoski

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 11.03.2016

Thesis supervisor:

Prof. Raimo Kantola

Thesis advisors:

Ph.D. Corrado Leita

Ph.D. Petros Elia

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Luukas Larinkoski

Title: Detecting Encrypted Command & Control Channels with Network Fingerprints

Date: 11.03.2016 Language: English Number of pages: 5+76

Department of Communications and Networking

Professorship: Communications Networking

Supervisor: Prof. Raimo Kantola

Advisors: Ph.D. Corrado Leita, Ph.D. Petros Elia

The threat landscape of the Internet has evolved drastically into an environment where malware are increasingly developed by financially motivated cybercriminal groups who mirror legitimate businesses in their structure and processes. These groups develop sophisticated malware with the aim of transforming persistent control over large numbers of infected machines into profit. Recent developments have shown that malware authors seek to hide their Command and Control channels by implementing custom application layer protocols and using custom encryption algorithms. This technique effectively thwarts conventional pattern-based detection mechanisms.

This thesis presents network fingerprints, a novel way of performing network-based detection of encrypted Command and Control channels. The goal of the work was to produce a proof of concept system that is able to generate accurate and reliable network signatures for this purpose. The thesis presents and explains the individual phases of an analysis pipeline that was built to process and analyze malware network traffic and to produce network fingerprint signatures.

The analysis system was used to generate network fingerprints that were deployed to an intrusion detection system in real-world networks for a test period of 17 days. The experimental phase produced 71 true positive detections and 9 false positive detections, and therefore proved that the established technique is capable of performing detection of targeted encrypted Command and Control channels. Furthermore, the effects on the performance of the underlying intrusion detection system were measured. These results showed that network fingerprints induce an increase of 2-9% to the packet loss and a small increase to the overall computational load of the intrusion detection system.

Keywords: malware, intrusion detection system, command and control channel, network fingerprint

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Luukas Larinkoski

Työn nimi: Salattujen komento- ja ohjauskanavien havaitseminen verkkosormenjälkien avulla

Päivämäärä: 11.03.2016 Kieli: Englanti Sivumäärä: 5+76

Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoverkkotekniikka

Työn valvoja: Prof. Raimo Kantola

Työn ohjaajat: TkT Corrado Leita, TkT Petros Elia

Internetin uhkaympäristön radikaalin kehittymisen myötä edistyksellisiä haittaohjelmia kehittävät kyberrikollisryhmät ovat muuttuneet järjestäytyneiksi ja taloudellista voittoa tavoitteleviksi organisaatioiksi. Nämä rakenteiltaan ja prosesseiltaan laillisia yrityksiä muistuttavat organisaatiot pyrkivät saastuttamaan suuria määriä tietokoneita ja saavuttamaan yhtämittaisen hallintakyvyn. Tutkimukset ovat osoittaneet, että tuntemattomien salausmenetelmien ja uusien sovellustason protokollien käyttö haittaohjelmien komento- ja hallintakanavien piilottamiseksi tietoverkoissa ovat kasvussa. Tämänkaltaiset tekniikat vaikeuttavat oleellisesti perinteisiä toistuviin kuvioihin perustuvia havaitsemismenetelmiä.

Tämä työ esittelee salattujen komento- ja hallintakanavien havaitsemiseen suunnitellun uuden konseptin, verkkosormenjäljet. Työn tavoitteena oli toteuttaa prototyyppijärjestelmä, joka analysoi ja prosessoi haittaohjelmaliikennettä, sekä kykenee tuottamaan tarkkoja ja tehokkaita haittaohjelmakohtaisia verkkosormenjälkitunnisteita. Työ selittää verkkosormenjälkien teorian ja käy yksityiskohtaisesti läpi kehitetyn järjestelmän eri osiot ja vaiheet.

Järjestelmästä tuotetut verkkosormenjäljet asennettiin 17 päiväksi oikeisiin tietoverkkoihin osaksi tunkeilijan havaitsemisjärjestelmää. Testijakso tuotti yhteensä 71 oikeaa haittaohjelmahavaintoa sekä 9 väärää havaintoa. Menetelmän käyttöönoton vaikutukset tunkeilijan havaitsemisjärjestelmän suorituskykyyn olivat 2-9% kasvu pakettihäviössä ja pieni nousu laskennallisessa kokonaiskuormituksessa. Tulokset osoittavat, että kehitetty järjestelmä kykenee onnistuneesti analysoimaan haittaohjelmaliikennettä sekä tuottamaan salattuja komento- ja hallintakanavia havaitsevia verkkosormenjälkiä.

Avainsanat: haittaohjelma, tunkeilijan havaitsemisjärjestelmä, komento- ja ohjauskanava, verkkosormenjälki

Contents

Abbreviations

API	Application Program Interface
APT	Advanced Persistent Threat
AV	Anti-Virus
C&C, C2	Command and Control
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DDoS	Distributed Denial of Service
DGA	Domain Generation Algorithm
DNS	Domain Name System
EK	Exploit Kit
FN	False Negative
FUSE	Full System Emulation
FP	False Positive
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IRC	Internet Relay Chat
JIT	Just-in-time
P2P	Peer-to-peer
PCAP	Packet Capture
PCRE	Perl Compatible Regular Expression
RAT	Remote Access Trojan
RFC	Request for Comments
SE	Social Engineering
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TP	True Positive
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine

1 Introduction

Today modern societies are dependent on digital information infrastructures to store, process and transmit data. Critical organs of these societies, such as government offices, industrial plants and power grids, are reliant on the availability and reliability of computer systems that connect and control crucial structures. These computer systems are part of what can be described as the global cyberspace. The proliferation of personal computers and portable smart devices has broadened the cyberspace to encompass all digital devices that communicate over networks. This cyberspace provides the environment for benevolent actors, e.g. individuals, companies and governments, to act as part of the society and to carry out everyday tasks to progress welfare socially and economically. However, at the same it also provides means for malicious actors to pursue their own agenda, which can take its form e.g. as cybercrime, espionage or terrorism. Direct or indirect connections to subsets of the cyberspace, such as the Internet, can expose systems to these cyberthreats imposed by the malicious actors.

Cyberthreats have been a known problem for a long time, but the threat landscape has evolved drastically in the past two decades. An important factor in this evolution is the diversification of the malicious actors. The stereotype of a hacker living in a basement and writing malware for the sake of fun, causing mischeaf or gaining "street credibility" is no longer in effect [1–3]. Frequent discoveries of security holes and vulnerabilities, which are an inevitable side effect of the digitalization of the society, have opened a lucrative opportunity for criminal groups to capitalize on. As a result, the actors responsible for recent notorious malware are often cybercriminal organizations that mirror legitimate businesses in their organizational structure and processes in the effort to reach maximal financial profit [2]. Targeting large masses of Internet users rather than isolated entities, such as companies or individuals, exposes the largest attack surface for attackers and hence any machine that is exposed by a vulnerability can be seen as a potential source for a malware infection. More organized and diverse malicious actors have thus increased the probabilitiy with which a normal user can become a target of a cyberattack on the Internet.

In non-targeted malware distribution campaigns the goal of a successful infection is often to subject the victims' machine under the control of the malicious entity. Such infected machines under the control of the same malicious entity are collectively called a *botnet* and the controlling entity the *botmaster* [1, 4]. Botnets can function as a multi purpose tool for cybercriminals to perform various tasks, such as sending spam emails, performing distributed denial of service (DDoS) attacks or stealing private information, e.g. banking information or passwords [1, 4–8]. "Botnet as a service" [1, 2] and similar schemes render it relatively simple for anyone to obtain partial or full control of a botnet. Attributing responsibility for attacks has therefore become problematic as anyone who is willing to spend as little as a few hundread dollars can become a malicious actor on the Internet [2, 8].

A general requirement for achieving prolonged control over an infected machine

is to establish and maintain a communication channel between the malicious actor and the victim machine. This channel, termed a Command and Control (C&C, C2) channel, can be used by the malicious entity to send commands or to transfer updates to the malware, and to download collected data from the infected machine [9]. Many malware utilize application-layer protocols, such as Hypertext Transfer Protocol (HTTP) or Internet Relay Chat (IRC), as end-to-end carrier protocols to establish this channel [10]. The practice of using standardized and well-defined higher-level protocols typically introduces invariants into the network traffic that through observation of consistent application by a malware can be used to identify and assign the malicious communication to the specific threat. Modern botnets however, such as Fynloski, Zeus P2P [11] and Virut, utilize lower-level non-descriptive carrier protocols, i.e. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), in their C&C channels, and obscure the carried higher-level C&C protocol using custom encryption algorithms. This reduces significantly the number of availabe invariants in the network traffic and renders reliable detection of C&C channels of modern botnets hard, if not impossible. [4, 12]

Network-based intrusion detection systems (IDS) are security appliances that monitor network traffic with the goal of detecting and intercepting malicious activity, such as C&C channels. IDS systems are versatile platforms that typically implement powerful functionality such as protocol dissection, flow reassembly, deep packet inspection and file extraction from passing traffic [13]. Detection is performed by applying special network signatures on the monitored traffic and raising alerts for matches. These network signatures define constraints and thresholds for malicious activity and are typically generated and distributed by professionals in the security industry. Depending on the nature of an event, the system can for example respond actively by blocking the traffic, or simply log the event and notify an administrator. Deploying an IDS system can therefore provide an effective base for protecting a network against various cyberthreats.

The inherent reliance of IDS systems on network signatures underlines the importance of maintaining a sufficiently large quantity and high quality signature base. The amount and quality of signatures directly impacts the accuracy of performed threat detection and thereby influences the level of achievable protection provided by an IDS system. As a consequence, implementing a scalable process and infrastructure that produces network signatures is an imperative requirement for any entity that makes use of, or provides, IDS solutions. The rapid evolution of the threat landscape necessitates a continuous development of these processes in order to produce new detection capabilities against emerging threats. Recently, the gradual shift towards encrypted malware communication and utilization of non-descriptive carrier protocols has induced a need for new and innovative responses.

This thesis explores one possible approach for the detection of covert C&C channels that employ custom encryption on top of non-descriptive carrier protocols. The work focuses in developing a proof of concept analysis and signature generation system that can be incorporated at the side of a commercial breach detection solution. The goal is to develop a framework that can produce network signatures that are capable of performing reliable and accurate detection against evasive malware without

compromising performance. This goal is approached by specifying heuristics for the detection of covert C&C channels, developing a feature extraction system that is able to correlate network traffic and cluster network flows based on a set of defined criteria, and translating this information into network signatures for the open source-based Suricata IDS.

This document is organized in the following manner. In the next chapter the theoretical background of Command and Control infrastructures of malware are explained. The chapter further discusses some commonly encountered evasion techniques that malware employ to thwart detection efforts. In addition, the second chapter presents the current state of the art and provides insights about the research that has been done in this field. The third chapter describes the context and methodologies of the work and defines the boundaries and expectations for the resulting system. Chapter 4 explains in detail the implementational specifics of the proof of concept signature generation framework that is presented in this thesis. The products of the system are deployed in real-world production environment networks to undergo qualitative and quantitative testing and verification. Chapter 5 presents the results of the tests and evaluates the performance achieved with the proof of concept system. The chapter furthermore discusses the results and the available test environment. Finally, Chapter 6 presents the conclusions of the work.

2 Command and Control channels in malware

Advanced malware is characterized by the requirement for a reliable and robust communication channel. The basic premise of such communication is that malware infected machines must be capable of using a set of pre-defined methods and channels to retrieve commands from the attacker and execute them. The ensemble of protocols, network topology and operating model that fulfils this premise is called the *Command and Control infrastructure*. At the simplest it comprises one or multiple servers and a connecting network that functions as the means to transmit data from the infected machine to the attacker. The C&C infrastucture can incorporate multiple C&C channels that are defined by protocols and message structure, and intermediary destinations used in the communication. [14, 15]

The establishment and use of Command and Control channels is an imperative step in recent attacks. It identifies the stage in the life cycle of an attack where the compromised system contacts back the attacker in order to receive commands and to relay information, such as status updates or collected data. To detect and identify malware C&C channels is therefore important from the defenders' point of view in preventing financial damage and theft of confidental data. The evolution of Command and Control techniques has been largely driven by defense efforts to uncover and detect malicious communication. This has developed into a constant battle between malware writers and security professionals, where the former repeatedly comes up with novel ways to perform C&C communication and evade detection efforts by the latter. [14]

This chapter explains concepts behind Command and Control infrastructures and how C&C channels are established in malware. The first section presents basic principles of a prevalent threat class that employs C&C communication in the core of its operations. The second section studies C&C channels from topological and functional point of view and covers implementation methodologies that are commonly employed by different malware. The fourth section illustrates different evasion and anti-detection techniques that are frequently used by current malware to achieve persistance and stealthiness. In the fifth section existing work is presented and some already researched approaches are discussed. The last section provides a short summary of the whole chapter.

2.1 Operational principles of Command and Control malware

The Internet of today exposes its users to a vast repertoire of rampant malware. Cyber-threats have been a noted problem for a long time, but the threat landscape of the Internet has evolved drastically in the past two decades. The earliest worms and viruses started to emerge in the Internet in the 1990s and since then the number of malware has continued to grow exponentially. Security vendors estimate that the

number of distinct samples has increased from about 10,000 in 1996 to hundreds of millions in 2016 [16–18]. Part of this rapid growth can be attributed to the usage of client and server polymorphism in malware, [16, 19] but part is also due to the shift in the threat environment: cyber-crime has gradually evolved into a lucrative business for criminals. This trend has introduced large groups of new players to the malware scene who produce novel and more sophisticated threats [2, 3].

Malware leveraging Command and Control communication have become the primary means for financially motivated cybercriminals to carry out their operations [1]. Well built C&C infrastructures offer versatile platforms that serve essential purposes in the coordination and realization of various malicious tasks. Common nefarious actions that feature C&C communication include activities such as sending spam, launching DDoS attacks, stealing personal data and espionage [1, 5, 20]. One general factor in the profit models of such activities is the continuity of the operation or campaign [21]. This is tightly linked with the structural decisions taken in the design phase that affect both the robustness and resilience of the C&C infrastructure against defenders' takedown efforts, and the properties that assure covertness against detection.

Botnets form one of the most serious threats on the Internet and exemplify well the types of malware that employ C&C infrastructures in their core. The term *botnet* is used to describe networks of infected machines that are controlled by a human operator, commonly known as the *botmaster*. A bots, i.e. the malware that infects a machine, can then be used to perform various tasks under the control of the botmaster. This process is illustrated in Figure 1, which depicts a typical life cycle of a botnet [15]. The first phase, the infection phase, is usually continuous throughout the existence of the botnet and overlaps with the other phases of the life cycle. In this phase new machines are infected either through direct actions by the botmaster or by indirect spreading efforts, e.g. by existing bots or other malware that installs the bot on a victim's machine. The second phase constitutes the commanding and controlling of the botnet after its establishment. This phase and its components are further discussed in the next section. Controlling the botnet over C&C channels allows the botnet controller to use the power of the collective infections and to exert it for malicious purposes, denoted by the third phase in the figure. [1, 4, 12, 15, 22]

The actors responsible for deploying and controlling bot networks are manifold and often unambiguous. Attracted by financial profit, the malware industry has seen an increase in organization of cybercriminal groups who mirror legitimate businesses in organizational structure and processes striving for high efficiency and returns [2]. In conformance with methods dictated by conventional business models, these criminal groups sell their products and offer consultant services for setting up and operating botnets thereby introducing new malicious actors to the scene [2]. Furthermore, toolkits, such as Zeus bot [11, 23, 24], that are designed to streamline and automate the process of creating a botnet, are distributed freely on Internet forums and have contributed to the build-up of the number of unique variants and responsible actors [25]. Malware produced by such toolkits often display indistinguishable characteristics and mask the actors behind a generic implementation of the common malware [24]. This commercialization and availability of sophisticated malware

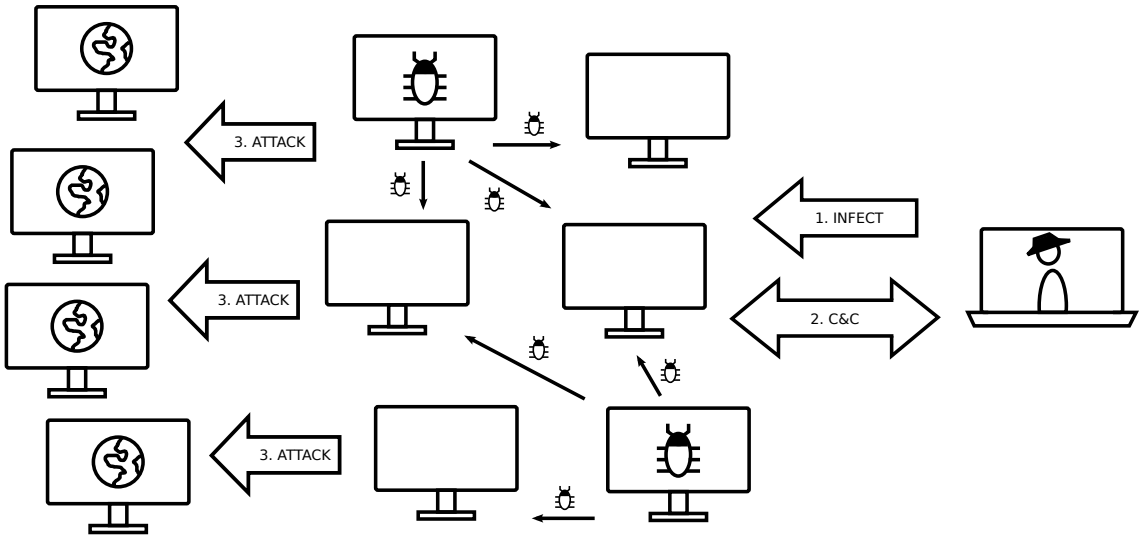


Figure 1: Botnet life cycle.

frameworks has caused a shift in the malware scene where creating a botnet has never been easier.

Due to the large number of malware and wide range of infection vectors, ending up infected with a bot is increasingly more common and often inconspicuous. Resulting from its broad use on the Internet, vulnerabilities in browsers are a major target for attackers. Software bugs in browser plugins, such as Java, Flash or Silverlight, are often subject to drive-by attacks, whose goal is to exploit an outdated and vulnerable version of the plugin in order to load and execute a malicious binary. This approach has been commercialized by cyber-criminals in the form of exploit kits (EK) that combine and put together a large number of targeted exploits against different versions of vulnerable browser plugins. Exploit kit developers offer infections as a service where a malicious actor pays for the kit to exploit a vulnerability on a machine and to install a malicious binary supplied by the actor. This effectively hands over the control of the machine to the malicious actor, who is often embodied by a botnet controller. These kind of drive-by attacks form a particularly prevalent threat to Internet users as getting infected only requires a visit to a website that hosts the malicious exploit script. [26]

Another significant source of infections is exploiting the human nature through social engineering (SE). Social engineering is a process in which a malicious actor convinces or tricks an unsuspecting user into performing an action that ends up profiting the malicious actor at the expense of the user or another victim. At the simplest the action can take its form in a phone call, where the caller impersonates another person and requests information or an action from the person answering the phone that would be authorized only to the real person. This principle can be applied to email. In these instances the attacker sends an email with a malicious attachment that upon opening exploits a vulnerability on the computer's software that opens it and installs a malware. These social engineering techniques are often employed in combination with spam emails ensuring large-scale distribution and

increased number of infections [11].

After a successful infection the bot may be ordered to perform a manifold of tasks. These tasks are predominantly driven by motives that can be split into three categories: financial profit, expansion of the botnet, and causing financial or operational harm to a target. Financial profit can be generated in multiple ways when operating a botnet. Commonly encountered examples include advertisement click fraud [27], harvesting and selling confidential information from infected machines [1], and sending spam emails [5]. [8] Botnet expansion can be sought for by e.g. attempting to infect machines located on the local networks of existing bots or sending spam emails with malicious attachments that infect recipients with a bot [11]. The third category includes actions that disrupt operations of a targeted individual or organization often causing financial damage and interruptions in service. The most common form of these attacks are DDoS attacks where a botnet is instructed to direct continuous network traffic to the victims network with the aim of saturating their routing systems and effectively rendering the network unusable [6, 20].

Botnets pose a diverse and prevalent threat to the Internet and its users. Due to the attractive financial possibilities of the malware business, new players are likely to keep emerging to the scene. However, botnets are not the only class of malware that both incorporate C&C infrastuctures and impose a serious threat. Advanced persistent threats (APT) are a more recent threat that are characterized by their persistence, high sophistication and targeted nature [14, 21]. Shaped by recent events and often motivated by industrial or (geo)political espionage and exfiltration of sensitive data, APTs represent efforts by highly-organized and dedicated cybercriminal groups [14, 21]. Despite the indisputable importance and interesting nature of APTs, for simplicity's sake the rest of this chapter focuses on presenting C&C infrastructures from the botnet's perspective. However, majority of the discussed principles apply also for APTs and are actively employed by them.

2.2 Command and Control topologies of botnets

The majority of botnet communication is built on well defined network protocols and uses existing Internet services, such as the Domain Name System (DNS), to implement their channels. Having an understanding of the protocols and services from both the attacker's and defender's point of view is key to understanding what the attackers are capable of doing, and building a strategy to detect and fight C&C malware. This strategy includes defining taxonimies that classify existing botnets with the goal of helping defenders to respond to future emerging threats [28, 29]. As a response to uninterrupted efforts by security professionals to detect and take down botnets, malware writers experiment with alternative strategies to build more robust and reliable C&C infrastuctures. [10, 14, 15]

With the shift towards an environment where malware is developed and spread for financial profit [2, 3], C&C topologies that are cheaper to build and maintain are favored over topologies that are more robust, but harder to implement. The earliest and most widely implemented design follows a centralized architecture where

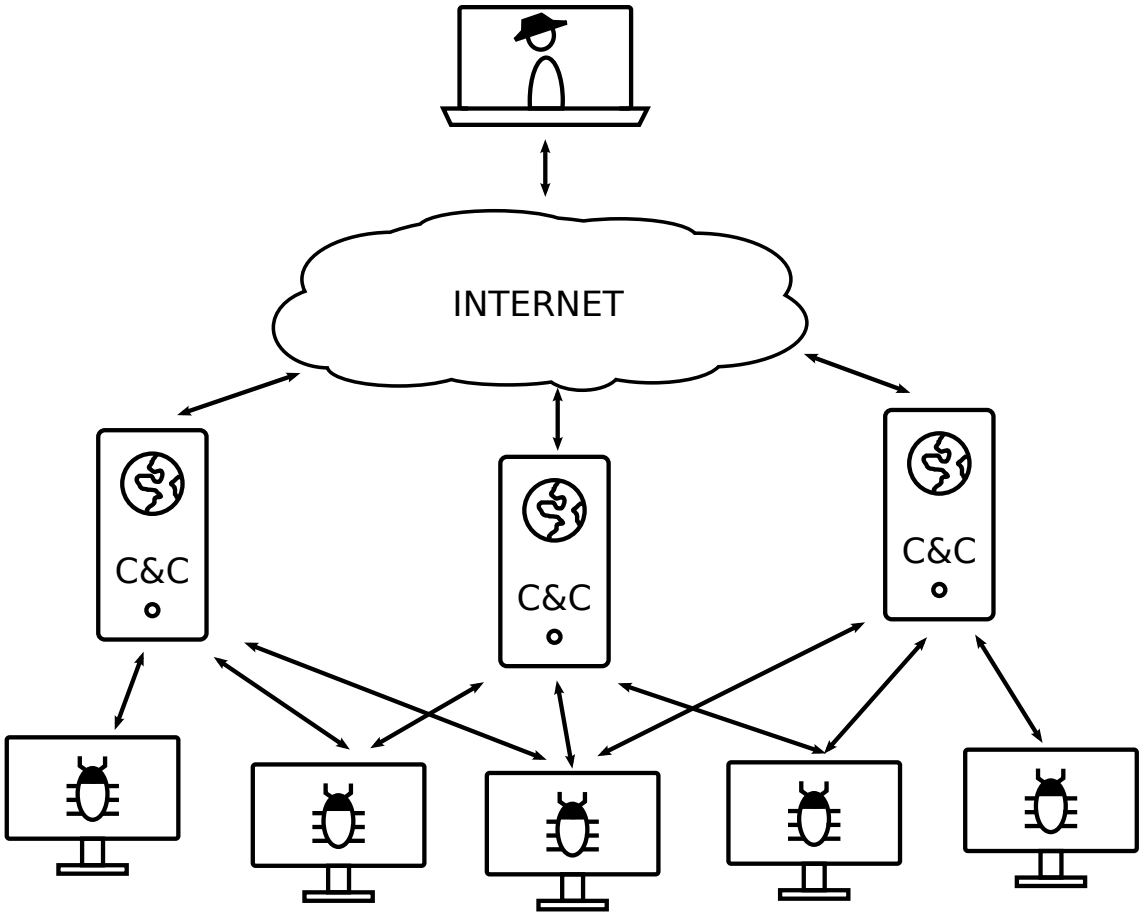


Figure 2: Centralized botnet architecture.

one or more network nodes function as C&C servers and are used exclusively to forward messages between bots and a botmaster [25]. When a victim is infected, the malware will connect to the C&C server to communicate its existence and to retrieve commands. The C&C servers, also called *rendez-vouz* points, typically run services like IRC or HTTP on pre-defined ports and serve incoming connections from the bots [1]. This client-server architecture provides a simple platform for the malware to get in contact with C&C servers and to transmit collected information back to the attacker. Similarly it's easy for the attacker to deliver new commands to the bots and to maintain the Command and Control architecture. [15]

An example of the centralized botnet topology is illustrated in Figure 2. Infected machines, depicted at the bottom of the figure, embed sufficient information about protocols, message structure and destinations in the malware code to figure out how to connect to a server in order to establish the C&C channel [30]. To cover their tracks an attacker can use compromised machines as C&C servers or servers from legitimate providers that are misused under false identity for malicious purposes. In addition, the connection between the botmaster and the C&C servers often includes multiple stepping stones, i.e. intermediary proxy nodes, whose function is to act as gateways for the communication in order to mask the real location of the botmaster [14]. The

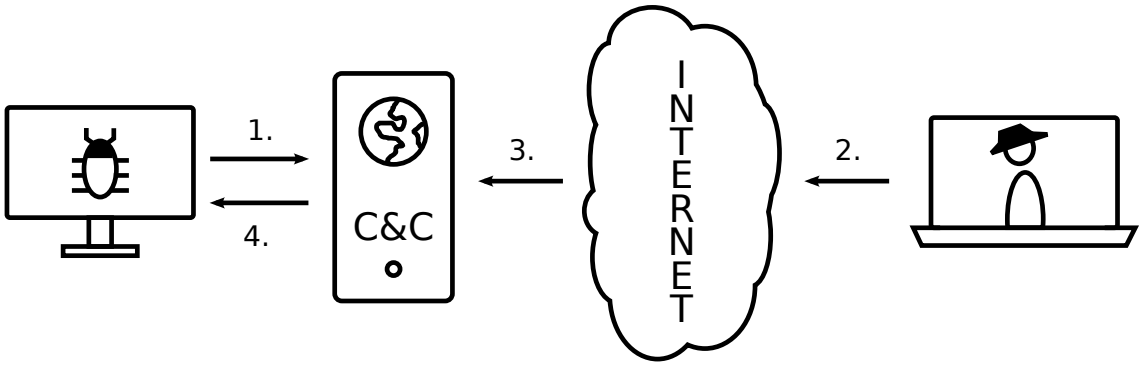


Figure 3: Push-based Command and Control channel.

use of stepping stones hampers significantly the traceback process to the attacker, and might require cooperation and legal actions in different countries. [31]

The centralized topology can be further categorized into two subclasses, *push* and *pull*, depending on how a botmaster's commands reach the bots. In a push style C&C, illustrated in Figure 3, the bots connect to a C&C server and wait passively for commands from the botmaster. The botmaster issues new commands on the C&C server and all bots that are connected to the server receive the commands in real-time and start executing them. This gives the botmaster real-time control over the botnet. After the bots have completed the tasks, they report back the result on the C&C server and wait for new commands. [4, 10, 14, 15]

The classic example of a push style centralized C&C makes use of an IRC server. IRC is a lightweight text chat protocol that provides chat rooms, or *channels*, for group conversations, and private user-to-user chats. Channels are hosted on IRC servers, which are part of IRC networks. Most channels on IRC networks are publicly available, but it is possible to require authentication with a password. This provides lightweight privacy and protection for botnet channels to hide on public IRC servers. The flexibility of the IRC protocol and the availability of mulitple open-source implementations provide a simple platform for malware to perform C&C. IRC clients are typically embedded in the malware code itself and don't require any pre-installed programs on the victim machine. [10, 14, 15]

In a pull style C&C, most often implemented through the HTTP protocol, the botmaster simply connects to a C&C webserver and publishes the commands in a file that is accessible by the bots. Upon infection, the bots connect to the C&C server with an Uniform Resource Locator (URL), such as `http://hostname/request?id=[6-byte-bot-id]&country=[2-character-country-code]`, to retrieve any commands placed passively by the botmaster. The file can be a server-side script that contains a logic to determine which command, or file, is returned given the characteristics of the connecting bot, e.g. bot identifier or residing country of the victim machine. An example is demonstrated in Figure 4, where the botmaster places a command file on the C&C server and a bot later retrieves the file with an HTTP request. In the pull style approach the botmaster doesn't have real-time control over the bots, because there is a delay between the time when the botmaster issues a command

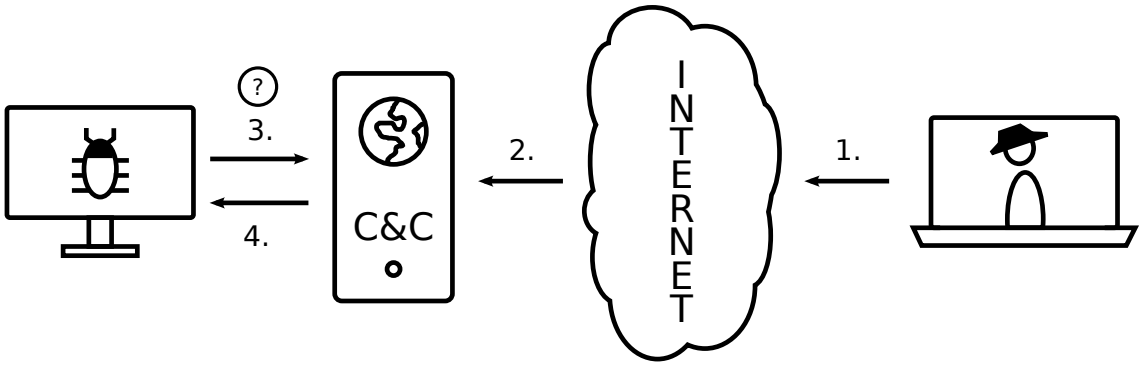


Figure 4: Pull-based Command and Control channel.

and when the bots check for new commands. The pull style is therefore relatively loose, i.e. the propagation of commands to the botnet has higher latency. [10, 15]

Centralized topoligies provide a cheap and straightforward C&C architecture for botnet operators. C&C protocols based on IRC and HTTP are easily customizable to suit the needs of a particular operation rendering consitent pattern-based detection difficult. Changes to the botnet structure, such as new C&C servers, are simple to introduce as the information can be propagated quickly across the botnet. Defender's efforts are complicated by these low delay adapting capabilities of centralized botnets. However, centralized C&C networks are not scalable and controlling hundreds of thousands of bots requires careful coordination amongst a large number of C&C servers. Large traffic volumes to central points of the network also induce an increased risk of detection through traffic monitoring. Centralized C&C servers thus expose the weakest spot of a botnet and offer few points of failure whose takedown can effectively destroy a botnet. [10, 14]

To overcome these structural limitations and scalability issues of centralized architecturess, many malware writers have migrated to distributed and decentralized designs. Distributed botnet topologies are characterized by the lack of a hierarchical structure and central servers that would solely provide a channel between the bots and the botmaster. Instead each node is ranked as hierarchically equal in the network and can act as both client and server to other nodes. Distributing the C&C communication over a sufficiently large group of interconnected nodes counters the issue of scalability as none of the nodes is used as the only gateway for the collectively generated traffic. [14, 15, 32, 33]

The decentralized design is influenced by various peer-to-peer (P2P) file sharing applications and protocols, such as Napster, Kazaa, Gnutella and Bittorrent. To illustrate this connection, consider the following example of the Bittorent sharing network. When a user wishes to download a file from the Bittorrent network, the Bittorrent client queries a central server for a "torrent" file, i.e. a tracker file that contains metadata about the searched file. The actual data file is virtually split into multiple pieces where each piece can be downloaded from several locations on the network. Based on the metadata stored in the torrent file, a tracking server coordinates the choice of peers that are involved in sharing pieces of the file and

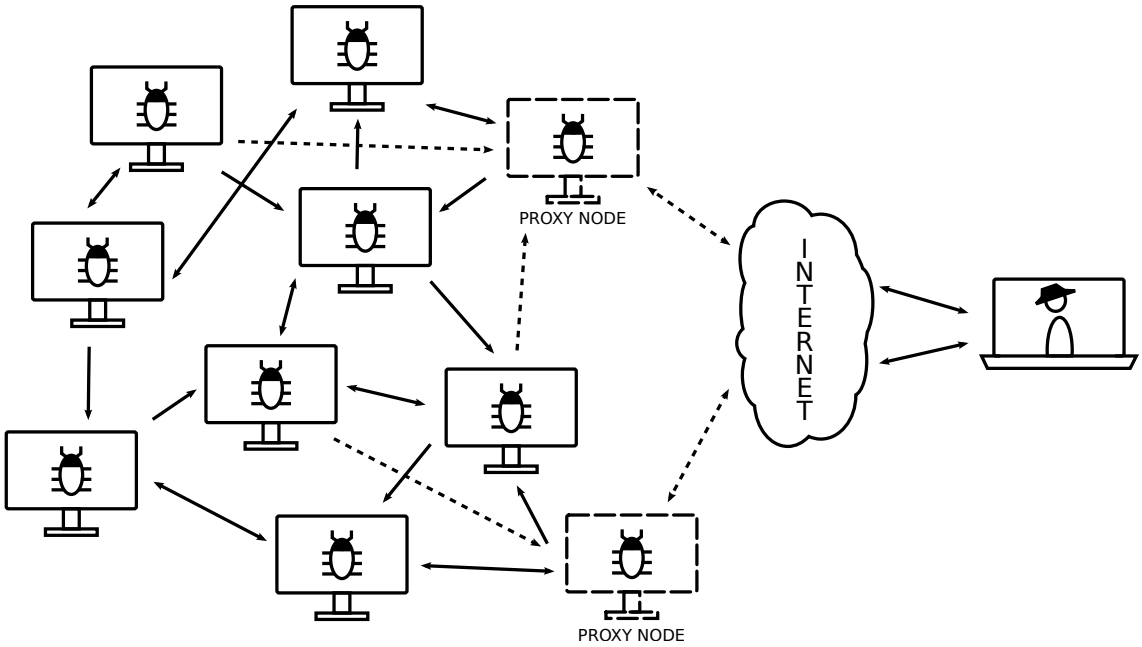


Figure 5: Decentralized botnet architecture.

enumerates these peers into a list. The user connects directly to individual peers from the list and fetches separate pieces until the whole file has been completely downloaded. While downloading the file, the user node itself is involved in sharing in that it offers already downloaded pieces to other nodes in the network that are in the process of downloading the same file. This effectively produces a balanced sharing network where information is propagated in a distributed manner from peer-to-peer.

The same premise is implemented in decentralized botnets with certain differences. Instead of a central server that coordinates peers to a given node, each node comes with a precompiled preliminary list of peers that exists in the malware code at infection time. For a new node in a bot network, this list provides the first immediate peers with whose help the node becomes part of the botnet. When a new machine is infected, the malware engages in information exchange with its immediate peers in order to update its peer list, and in order for the peers to propagate the existence of the new node in the network to their respective peers. In this manner new nodes are quickly integrated to the botnet without requiring a central server. [14, 15, 25]

The decentralized botnet architecture is illustrated in Figure 5. Each of the bots in the botnet constrain their communication to a limited number of peers. This is essential for load balancing the traffic between nodes and is implementation specific to a malware. To map each peer to an approximate number of other peers and to distribute the load in the network, the information distribution between two adjacent peers may not be bidiretional. This is denoted in the figure with directed arrows between infected machines. When the attacker wants to issue a command to the botnet, it injects the command to a node or a group of nodes. In push-based propagation the bots receiving the commands inform their peers of the newly supplied command. By flooding the information to each node's chosen peers,

the message is distributed further in the botnet until it has propagated the whole network. Alternatively in a pull-based propagation method each node queries its peers periodically for new commands. [14, 15, 33]

Decentralized botnets expose a very small number of crucial weak spots that might lead back to the attacker. In comparison to the centralized model, where the C&C servers constitute a definite link between the attacker and the data that is distributed through them, the decentralized model exposes at minimum only one entry point from the attacker to the entire bot network. As each of the decentralized nodes simply relay information that they receive from their peers, the connection between the distributed data and how the data was acquired are meaningless for the defenders. This operational model ensures that the trail to the entry point from an arbitrary observation point is quickly dissipated and nearly impossible to trace back. Furthermore, as in the case of centralized topologies, the attacker may route its connection to the entry point through several stepping stones further hampering the traceback. [14, 15]

The lack of centrally coordinated C&C communication offers a distributed botnet robustness and greater resilience against disruptions compared to for centralized networks. Decentralizing the C&C structure, and the associated C&C channels, to a distributed network removes by definition any focal frailties from the architecture whose takedown might disrupt the operation of the botnet. Maintaining a list of peers provides redundancy and strong guarantees of availability for the network. If for example nodes in the network are taken offline for some reason, such as failures or seizures, the gaps in the network are closed through intercommunication between bots and the botnet continues to operate under the control of the attacker. Dismantling a decentralized bonet can therefore require substantial efforts and disconnecting large portions of the network. [14, 15, 32]

Although the distributed model offers various design characteristics that are attractive to malware writers, the popularity and adoption rate of such botnet architectures has not yet caught up with centralized botnets. This might result from the difficulty of designing and implementing a protocol that is able to reach necessary requirements to achieve the properties discussed in this section. The effect is also largely influenced by the lower implementation cost and simplicity of the centralized architectures that might provide a greater return of investment on a shorter timescale. Nevertheless, due to the endless efforts by security professionals to dismantle botnets, malware writers may migrate increasingly to more resilient botnet topologies.

2.3 Common evasion techniques

The ongoing struggle to cripple and take down rampant botnets has lead to developments on the malware authors' side. Due to the inflexibilities in the formal processes of physically seizing C&C servers and thereby disrupting botnets, security professionals and researchers have shifted the focus into promoting more secure practices and developing services that offer privacy and security to their users. In response and encouraged by the defenders' efforts to prevent and detect botnet

communication, C&C designers repeatedly come up with innovative and novel ways to implement C&C channels. This section presents some techniques and obfuscation schemes that have been employed by malware authors in their C&C infrastructures in recent malware.

Compelled by strict security policies that block unused ports and services, many C&C protocols are tunneled through legitimate services [14]. While enforcing better practices, these policies have pushed C&C channels to be more frequently routed through well-known ports, e.g. 25, 80 or 443, effectively bypassing firewall restrictions and blending C&C communication into benign network traffic [14, 15]. To amplify the effect and to appear innocent, malware may create legitimate looking fake network traffic, e.g. by fetching content heavy popular websites, performing search engine queries or downloading files from image sharing sites. The increased noise in the traffic in turn slows down manual network traffic analysis by complicating the identification of the C&C channels.

The Domain Name System is an Internet service that maps easily memorizeable domain names to unique IP addresses [34]. In addition to benign uses, the DNS system allows attackers to direct C&C traffic to different destinations just by changing the underlying IP address of domain, providing flexibility and fault tolerance e.g. in case a physical server is seized by authorities. The DNS server responsible for a domain can set a maximum value for the amount of time, called time to live (TTL), that a returned DNS record can be cached until it must be resolved again. A technique utilized by attackers, called *fast-fluxing*, combines unusually short TTL values, often less than five minutes, with a rotating set of IP addresses that are included in each DNS record. As a result multiple IP addresses are associated for short periods of time to a single domain name. In fast-flux networks these IP addresses belong to compromised hosts that operate as proxy nodes and are configured to relay incoming traffic to central locations, often called *motherships*. This process is illustrated in Figure 6. In the figure, each of the bots on the left side performs a DNS query from the attacker controlled DNS server for the domain `attacker.com` and receives a list of IP addresses. The bots then engage in C&C communication with the proxy nodes that relay all traffic to a mothership server and to the attacker, who in turn may respond with instructions over the same channel. [35]

Fast-fluxing is resistant against endpoint takedowns and provides desireable properties for botnet controllers such as load balancing and protection against detection. However, security professionals have reacted to fast-flux networks by targeting its one weakness: the use of a single domain name [1]. Blocking access to known malicious fast-flux domains is an effective way to cripple C&C channels that depend on the availability of the domain. Conventions employed to achieve this include co-operating with domain registrars to suspend domain names, and through techniques such as blacklisting and domain sinkholing [1]. Domain blacklisting refers to a practice, where an authority in a network maintains a list of known malicious domains and compares each outgoing request with the list [14]. If a connection with a matching domain is found, either an alert is raised or the connection is blocked. Domain sinkholing on the other hand refers to the combined efforts of security professionals and DNS server authorities in which a malicious domain is shut out

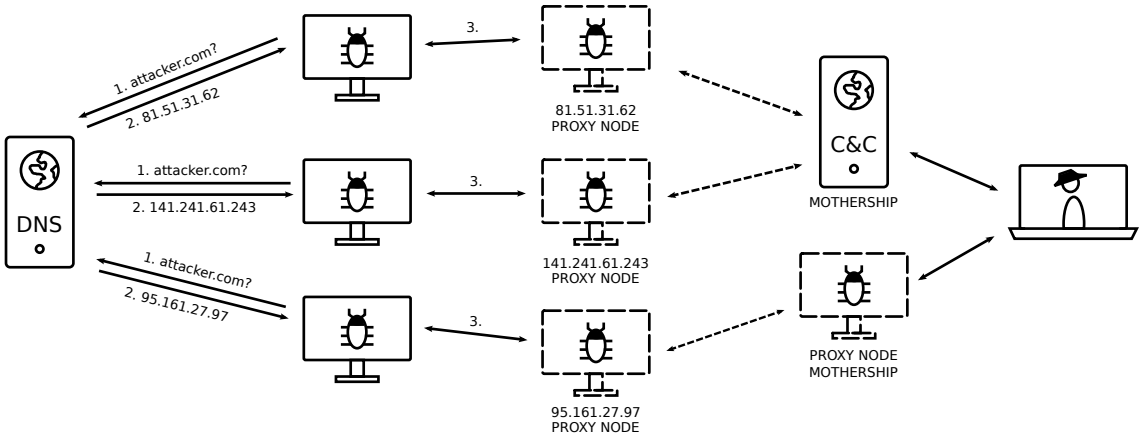


Figure 6: Fast-flux network.

from all external access. This is achieved by reconfiguring the DNS server that is responsible for resolving addresses of the domain under which malicious domain names have been registered, and instructing it to hand out safe IP addresses.

In response to these simple and responsive techniques to inhibit C&C communication to malware author's domains, C&Cs designers have introduced new styles to exploit features of the DNS system. This has taken its form in algorithms that dynamically derive rendez-vous points, which is most often achieved using domain generation algorithms (DGA). A DGA is an algorithm that takes a random seed as input and computes a list of domain names that can potentially lead to active C&C servers. DGAs are run periodically, e.g. once a day, using a seed that can be based on details such as the current date or time, word lists, Twitter trends or even foreign exchange rates [36]. The output of the seeded DGA algorithm is a list of domain names that form a set of potential C&C destinations. In order to connect with malware bots, the attacker uses the same seed to produce an equivalent list and registers one or more of these domains in order to set up rendez-vous points for the bots. If part of the domains or IP addresses get identified and taken down by defenders, the bots will keep resolving domain names until a domain succesfully resolves to a relocated C&C server. DGAs thus offer an agile strategy to provide redundancy against DNS-based defense techniques and to complicate C&C channel takedowns. [1, 14, 30]

In addition to escaping takedowns, malware authors seek to conceal the data being transmitted over C&C channels. Steganography is a practice of hiding a message in plain sight in such a way that it doesn't raise suspicions in onlookers or untrained eyes. Steganography provides an unobservable communication channel that can be employed using legitimate content, e.g. by encoding the messages inside redundant bits in image files. A real example of this is given in Figure 7, which portrays an image file that was downloaded by a malware over a C&C channel. When viewing the image in a usual way, it would display an animated running character. However, the image conceals a malware component that is only extractable by the malware author and the recipient, which in this case was a bot. This way malware can take

Figure 7: Hidden executable inside an image file.

advantage of the unobservability brought by steganographic means to perform hidden C&C communication. [14]

In order to evade pattern-based detection mechanisms, malware authors have increasingly employed even simpler techniques to ensure both unobservability and message concealment. Data obfuscation, e.g. through encoding or encryption, aims to remove cleartext information and invariants from the transmitted messages. Recently C&C designers have started implementing C&C protocols directly on top of lower-level carrier protocols, i.e. TCP and UDP. By encrypting, or otherwise obfuscating the payload, deriving the underlying plaintext is hard and at best impossible without knowing the encryption key or algorithm used for the obfuscation. If in addition the encryption key is secret and not easily retrievable, these C&C protocols effectively evade most conventional pattern-based detection systems. [12, 14]

2.4 Previous work on Command and Control detection

The problem of malware Command and Control communication and the associated practices of performing network-level detection are by no means new fields of research. A wide range of approaches have been studied and multiple different systems have been proposed. These continuous efforts are encouraged by the fact that C&C channels are often the weakest link in the operation of advanced malware. From the defender's point of view, C&C channels thus offer a suitable target and opportunity to effectively disrupt the functions of the complete malware C&C infrastructures.

This section presents previous research efforts that have been done to achieve this. In particular, proposed detection systems, which target C&C channels and thereby introduce relevance to the context of this thesis, are focused on.

Being the most commonly utilized way of performing C&C communication, centralized C&C infrastructures that are implemented with HTTP or IRC as carrier protocols are a self-explanatory target of detection efforts. Motivated by a need of good quality inputs for algorithms that automatically generate network signatures, Perdisci et al. [19] propose a system that is capable of identifying similarities in malware network communication. The analysis phase of the system tracks the way in which a malicious program interacts with the Internet and identifies distinct structural characteristics in the malware's network traffic. Similarities between samples are exposed by performing hierarchical clustering on the sequences of HTTP requests, which then serve as input to an algorithm that extracts network behaviour models for each cluster. These behaviour models represent the similarty between the samples in the cluster and describe the identifying features of C&C traffic. The models are entered into a network signature generation algorithm that produces Snort intrusion detection system (IDS) rules. Furthermore the generated network signatures are deployed in a real-world proof of concept experiment that verifies the overall feasibility of the detection system.

Similar approaches to this include efforts by Gu et al. [37] in BotHunter, which is a passive perimeter monitoring and bot detection system that relies on a pre-defined general infection model for botnets. Specific hosts that belong to a botnet are detected by performing *dialog correlation* between alerts raised by anomaly-based intrusion detection and the defined model. BotHunter is capable of detecting infected hosts regardless of an underlying C&C infrastructure or network protocol, given that the traffic generated by the bot strictly follows the user-defined infection model. BotSniffer [10], by Gu et al., on the other hand moves away from vertical correlation, i.e. host specific detection, and instead targets centralized botnet architectures through horizontal correlation. BotSniffer scans network traffic for behaviour that triggers anomaly-based signatures. Hosts in the monitored network that produce anomalous traffic are grouped by their network destination attributes, namely destination IP address and port. Hosts in these groups are then analysed for spatial-temporal correlation and similarity in order to verify features that are typical to centralized botnet architectures.

To overcome the limitations of being constrained to specific features in botnet C&C protocols and structures, Gu et al. have also proposed BotMiner [38]. BotMiner takes a similar approach as BotSniffer in that clients in the monitored network are clustered based on their traffic destination attributes, but also source attributes. This clustering happens on the so called *C-plane*, i.e. scope where the information about who is communicating with whom is determined. This clustering is, however, performed in general on the network traffic and doesn't imply maliciousness of a host by itself. In order to gain insight about malicious activities, BotMiner also operates on the *A-plane*, i.e. activity plane, where malicious activities in the traffic, e.g. spam, network scanning or binary downloads, are identified and used as a basis for further clustering. The information from these planes is combined using cross-plane

correlation, which produces indications whether a given host is likely to be part of a botnet. In this manner the detection system is not constrained to only C&C channels that show distinct features, but targets in general the inherent malicious features of botnet C&C communication.

All of the approaches presented above address threats that employ cleartext protocols in their C&C communication. Necessitated by the observed increase in obscured C&C protocols, research efforts that target encrypted C&C channels are a crucial requirement in order to detect prevalent new threats. ProVeX [12], by Rossow et al., is an approach that aims at detecting encrypted C&C traffic that is implemented on top of non-descriptive carrier protocols and thus doesn't expose any characteristic invariants in the traffic. The approach taken by ProVeX relies on the decryption of network traffic and n-gram-based malware classification. The a priori knowledge of the decryption routines and the capability to decrypt network traffic is obtained through reverse engineering the encryption algorithms of several recent malware. The decrypted plaintext C&C communication is then used to build probabilistic n-gram-based models for each malware. For this, byte positions in C&C messages are correlated over the communication of a malware family. The obtained probabilistic models are finally applied and matched on traffic that is being decrypted with all decryption routines on a network traffic sensor. ProVeX demonstrates that a decryption-based intrusion detection system is a feasible approach even on high-speed links, but acknowledges at the same time critical short comings, e.g. unlikelyhood of good scalability and the reliance on static encryption routines in malware.

Another approach against encrypted C&C traffic is CoCoSpot proposed by Dietrich et al. [39]. Unlike with ProVeX, CoCoSpot doesn't require a priori knowledge about the underlying plaintext message of the C&C communication or the associated C&C infrastructure. It's main contribution is a system that is capable of recognizing C&C channels based on known malicious communication behaviour. The recognition of these channels is based on the traffic analysis properties of transmitted messages that infer details about the structure of the C&C protocol. These properties are defined to comprise individual message lengths carried in HTTP, TCP, or UDP packets, and sequences of such message lengths. To train their traffic classifier, Dietrich et al. utilize a large set of malware generated network traffic that are obtained through malware executions in a controlled environment. The classifier is further supplied threat information through verified label information provided by an AV vendor. CoCoSpot argues that the classifier is capable of recognizing malicious C&C channels of a variety of recent botnets, and to attribute a threat for the observed traffic behaviour.

DISCLOSURE by Bilge et al. [40] takes this a step further by presenting a large-scale botnet detection system that doesn't require the transmitted payloads at all in detection the process. Instead, DISCLOSURE performs detection on C&C servers through NetFlow analysis that leverages patterns and flow characteristics that are typical to bots. In particular, the presented system extracts side-channel information regarding flow sizes, temporal behaviour and client-access patterns, and derives detection models that describe bot behaviour towards a C&C server. Bilge et al. utilize random forest classifiers to train their classifier and utilize ground truth labels

obtained from a third party threat intelligence company. The challenge that was faced when building DISCLOSURE was the false positive (FP) ratio that was induced by the derived criteria when matching on immensely large data sets. To reduce their FP ratio, Bilge et al. incorporated combined reputation score information from three external sources, including Google Safe Browsing, to facilitate the system's decisions on detections. The established DISCLOSURE system was, according to the authors, able to perform real-time detection of C&C servers with a false positive rate of 1 %.

2.5 Summary

Advanced malware that employ C&C infrastructures have become a large and prevalent threat on the Internet. Due to the large attack surface caused by a number of vulnerabilities in common software, ending up as a victim of a malware infection is a common peril. Efforts by security professionals have driven C&C designers to evolve and come up with novel and innovative ways to evade detection and thwart takedown attempts. Combined with the recent increase in encrypted communication, these sophisticated and persistent threats put the information security industry up for a real challenge. Thanks to the active field of study, new approaches are constantly tested and proposed. Thus, through combined efforts, smarter detection systems and developing better practices, these threats can be defended against. In the next chapter, the concept and underlying methodologies of a new analysis and detection system are proposed.

3 Context and methodolgies

In this chapter a concept is proposed that enables the detection of encrypted Command and Control channels. Specifically, encrypted C&C channels that are implemented on top of non-descriptive carrier protocols are targeted, as such communication often evades conventional signature-based network detection methods. The system leverages exposed side-channel information of C&C communication and produces signatures that constitute a new technique, called *network fingerprints*. These network fingerprint signatures are designed to be deployed on network-based intrusion detection systems monitoring the traffic at the edge between two networks. The presented system comprises multiple phases that are described in the following section and later in more detail in Chapter 4. The first section presents the motivation for the work by providing a description of the issues with existing detection systems with regards to encrypted C&C communication. The section proceeds to explain the scope of the work and lays the theoretical foundations for network fingerprints. The second section provides an overview of the architectural context in which this work is realized and introduces the components of the final analysis system that is established in this research. The third section states the main objectives of the work and clarifies the subgoals that are necessitated by the result. The last section provides a short summary of the whole chapter.

3.1 Problem statement and conceptual overview

A general approach to detect malware C&C channels are network-based intrusion detection systems. These systems commonly perform detection that is based on network signatures. Network signatures are a general name for a set of definitions, which dictate the means of how malicious network traffic is detected on a network and how detection events are handled. Commonly used IDS systems, such as Snort[1] and Suricata[2], provide flexible signature languages that enable precise ways to set these definitions. Traffic detection is usually realized in the form of boundaries and content conditions that, when fulfilled, trigger a detection event. Content matching is performed on the traffic through pattern matching methods, such as applying regular expressions. In the case of triggered detection events, special rules define matters such as how and when the IDS, and in the end an administrator of the IDS, should be notified about the events. These rules can include details such as thresholds, alert limits, time intervals and signature chaining, and also the output channel, e.g. logs.

Network signatures are generated by automated systems or through manual analysis, and target features in the C&C communication that are characteristic to a malware. These features are often found either in the details of the carrier protocol or in the carrier protocol's payload, or both. The *carrier protocol* is the

[1] https://www.snort.org/
[2] http://suricata-ids.org/

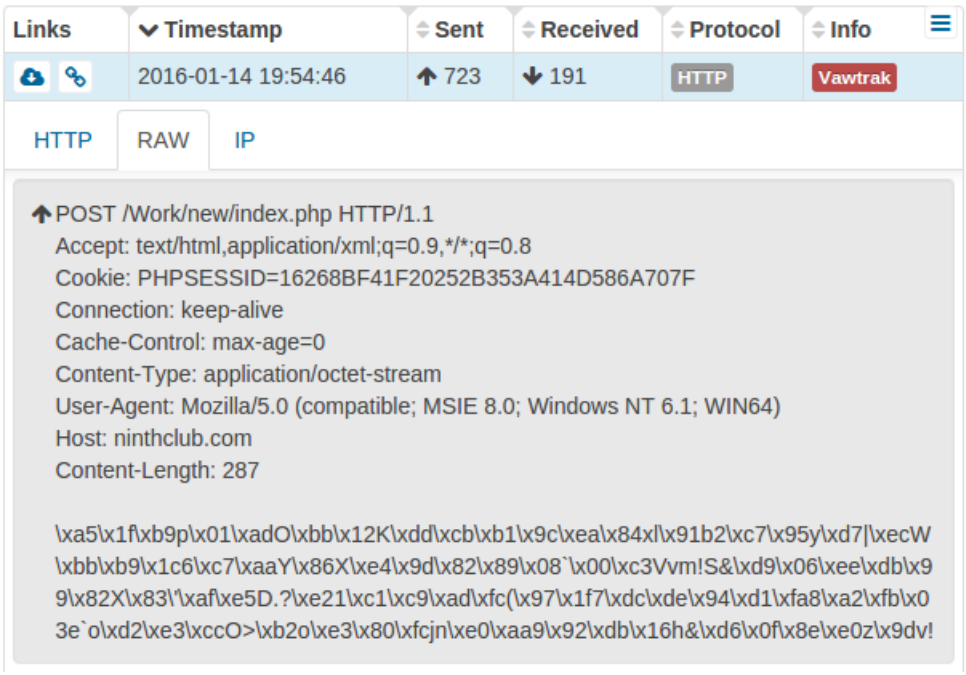


Figure 8: Descriptive carrier protocols exhibit tags and spatial delimiters.

underlying protocol that is used as a platform to transmit the actual C&C protocol. This can be an application-layer protocol, e.g. HTTP or IRC, or alternatively if not present, the transport-layer protocol, e.g. TCP or UDP, that is used to carry the application-layer C&C protocol. Most systems that perform network-based detection on C&C communication, including the ones presented in Section 2.4, require at least some a priori knowledge of the cleartext C&C communication. *Cleartext* in this context is used to denote communication that directly exposes structural patterns of the carrier protocol or the carried C&C protocol itself. Cleartext can therefore include artifacts, such as human-readable plaintext strings or spatial structures that encompass information, which can be useful for the C&C protocol parsers used by the bot and the botmaster.

The a priori knowledge of the internals of a C&C protocol present a significant advantage to detection systems. Protocols in general define the conventions and procedures that two parties can employ in order to exchange messages in a fashion that both parties understand the communication. This definition implies that the protocol includes defined invariants that are used to tag or delimit parts of the protocol messages. The existence of such static tags or spatial patterns can be therefore leveraged in detecting the protocols existence in the communication without the requirement of knowing the carried message. This principle is what most conventional signature-based network detection systems utilize in order to detect malware communication. These protocol invariants can be obtained directly from the messages if the traffic generated by a malware is not obfuscated. If the messages are encrypted or otherwise obscured, the invariants can be derived dynamically by e.g. decrypting the messages upon receiving, as was done by Rossow et al. in [12].

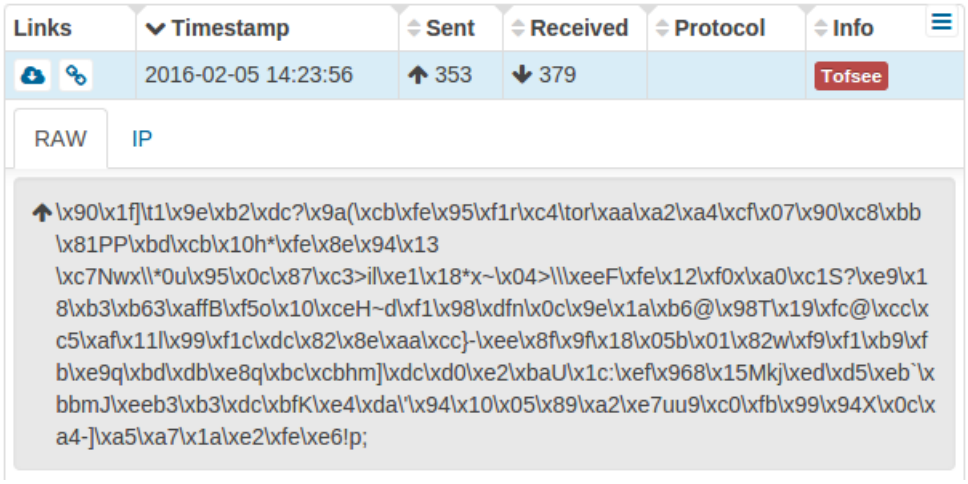


Figure 9: Encrypted payloads on non-descriptive carrier protocols don't exhibit any useful invariants.

Most current malware utilize carrier protocols that expose invariants to the C&C communication. Common application-layer protocols, e.g. HTTP and IRC, exhibit characteristic strings in their protocol headers regardless of the purpose of use and can be used as a basis for detection. In the context of this work, carrier protocols that demonstrate this feature are called *descriptive* protocols. An example of this is shown in Figure 8, where a C&C protocol transmits a seemingly obfuscated payload inside an HTTP protocol packet. The descriptive carrier protocol incorporates multiple characteristic delimiters and tags that can theoretically be used as a basis for identification. Perdisci et al. have shown that the identification of such invariants in malware generated HTTP traffic can be automatized and used in the generation of IDS signatures [19]. However, many prevalent C&C malware implement their C&C protocol on top of lower-layer carrier protocols, such as TCP or UDP. These protocols don't exhibit useful carrier protocol-specific characteristics for signature-based detection and are therefore called *non-descriptive* protocols. Due to the lack of useful headers, tags or delimiters in the carrier protocol, invariant-based detection is possible only if the C&C protocol is in cleartext or encrypted with a known, or easily deriveable, static key. [12]

Recent developments suggest that malware authors have increasingly started to employ encrypted Command and Control channels on top of non-descriptive carrier prototcols. C&C designers are thus shifting towards techniques that don't expose any valuable cleartext artifacts and effectively evade most existing detection systems. This is illustrated in Figure 9, where a non-descriptive carrier protocol is used to deliver an encrypted message in the payload, which conceals the cleartext C&C protocol message. While it is possible to obtain necessary information of a C&C protocol's invariants through reverse engineering malware samples, detecting these C&C protocols through traffic decryption in IDS systems doesn't provide a scalable solution. New systems are therefore required to supplement current invariant-based detection systems in order to maintain coverage of evasive emerging threats. [12]

This master's thesis presents a novel technique for the detection of encrypted malware C&C channels. The presented technique, called *network fingerprints*, specifically targets C&C channels that are instrumented to use non-descriptive carrier protocols and thereby would evade conventional network-based detection systems, as previously described. Along with the technique itself, a full-fledged analysis architecture is developed on the side, which performs automatized analysis and clustering on captured malware network traffic, and supports the generation of network fingerprint signatures for an intrusion detection system. The internals of this architecture are explained more in depth in Section 3.2 and Chapter 4.

Network fingerprints are a special type of network signatures that leverage side-channel vulnerabilities in C&C communication. Side-channel vulnerabilities in this context denote sources of information that may provide an eavesdropper who has access to the messages any insights about the content of the exchanged traffic. The term is often employed in conjunction with encrypted communication as cleartext traffic already exhibits exploitable characteristics by default. These side-channel leaks can be a cause of weak implementations or incorrect use of cryptographic algorithms, and can reveal useful information through the statefulness of an application, temporally aligned communication patterns, or packet characteristics, such as packet sizes and payload entropy. By observing vulnerable traffic that exhibits side-channel leaks, the observer can search for correlating features in a sufficiently large dataset and create accurate fingerprints of the incorporated interrelationships in order to later identify similar traffic. Identifying and leveraging side-channel leaks in C&C protocols thus enables an effective way of detecting encrypted C&C channels. [41]

One important requirement for the presented network fingerprint solution is the ability to perform reliable C&C channel detection on high-speed network links without introducing performance and packet loss to the underlying intrusion detection system. In order to achieve high scalability, constraints have to be set on the number and type of side-channel features that are targeted in the design of the new detection capability. A significant aspect of scalability in an intrusion detection system is its memory efficiency. Memory problems can be reduced for example by implementing features that don't depend on complex state computations, i.e. storing and operating on states, or require only a small state buffer. In order to meet these requirements, the dimensions of the targeted side-channel vulnerabilities have to be limited to a level that is still able to ensure a high detection reliability against encrypted C&C channels. Due to their straightforward measureability and computability, this work focuses on two side-channel leaks: payload size and payload entropy.

Payload sizes and entropy expose revealing information about a packet and the malware that participates in its transmission. Similarly to benign network protocols, C&C protocols often include authentication routines in the beginning of the C&C interaction where the C&C server verifies the authenticity of the bot before relaying further information, e.g. commands. This kind of C&C protocol behaviour suggests that there exist certain message sequences that conform more often to a usual pattern than others. Tracking the payload sizes of potentially reoccurring message sequences provides a simple and straightforward solution for identifying patterns in malicious C&C channels. Furhtermore, as mentioned earlier,

unencrypted cleartext C&C messages often exhibit invariants to the communication, such as tags or delimiters. The existence of these invariants affects the entropy of a message that is encrypted and thereby exposes information about the contents of the message itself. Extracting payload entropy information and associating it with payload size sequences enables a reliable way of deriving fingerprints of C&C protocol communication. This combination thus offers a simple and lightweight approach for the detection.

The combination of payload sizes and entropies in association to a message sequence allows to express network fingerprints in a clear vectorized form:

$$netfp = \langle (l_{c1}, e_{c1}), \ldots, (l_{cn}, e_{cn}),\ (l_{s1}, e_{s1}), \ldots, (l_{sn}, e_{sn}) \rangle,$$

where l refers to the payload size (or length) of a captured packet and e is the Shannon entropy of the packet payload. The individual tuples, consisting of the packet size and entropy, are dependent on the index of a packet inside a traffic flow. The index n of a tuple is the upper bound limit for which matching is attempted. Furthermore the directionality of the network traffic is taken into account by distinguishing between source and destination originated packets, denoted in the model by c, for *client*, and s, for *server*.

Network fingerprints operate in the context of network traffic flows [42], which are defined to be comprised of single TCP connections from a source IP address and port, the client, to a destination IP address and port, the server. A traffic flow is further divided into transactions, i.e. single messages sent by the source host or the destination host, which are carried over the network inside TCP segments. For simplicitys sake, the terms packet and segment are used to describe the same concept in this thesis and are used interchangeably. An individual transaction in a traffic flow is described in the network fingerprint by a single tuple. This connection is made by marking the tuples with the index of the targeted transaction in the flow. The values of a tuple at index i thus define the conditions that have to be met by the transaction payload i in order for the tuple to match. A tuple may consist of both entropy and payload size constraints or alternatively only one, which for example enables matching on only short packets, where the Shannon entropy provides an unreliable value due to undersampling [43]. Network fingerprints can also be a combination of tuples that target both client and server transactions, or only one of them. Furthermore, the indexing of the tuples is flexible, i.e. the transaction index between two tuples must not necessarily be sequentially incremental, allowing a network fingerprint to have "gaps". This is useful for example in a case where the state machine of a C&C protocol is known. Using specific indexes in the fingerprint would then allow to target specific states in the execution of the C&C protocol.

The flexible structure of network fingerprints provides adaptibility to side-channel detection of C&C communication. After generating a network fingerprint signature, the fingerprint is placed on a network IDS, which attempts to match the fingerprint on monitored traffic flows. A detection event for a flow is defined to occur, when all tuples of a network fingerprint match on their respective transactions in the flow. Therefore the reliability of an individual network fingerprint is directly proportional to the quantity and quality of the constraints in the fingerprint. In the general

context of network intrusion detection the reliability of a network signature is directly linked to the degree with which it generates false positive detections when exposed to benign network traffic. Due to the difficulty of modeling C&C communication in a generic way with respect to benign network traffic, deriving a theoretical model for the criteria of a good network fingerprint is difficult. For this reason the set of requirements that define a reliable network fingerprint are determined experimentally in Chapter 5.

Network fingerprints enable a broadened detection capability of a wide range of C&C malware. Unlike many existing research approaches that target a particular malware C&C infrastructure, e.g. centralized or distributed/P2P, network fingerprints are not constrained to be effective only against specific C&C types. In addition, based on the eliminated requirement of posessing a priori knowledge of the cleartext C&C protocol, network fingerprints can be theoretically used against any C&C communication that exposes payload size and entropy side-channel leaks. As a result the variety of malware that fall inside the limits of this detection approach include next to botnets also other prevalent threats, e.g. ransomware and APTs.

The detection capability for a malware can, however, only be created if the malware exhibits sufficient intrinsic similarities between its samples. This statement creates an integral assumption that malware samples of a single variant, which belong to the same malware family, behave deterministically and conform to the same commands and communication patterns, i.e. cleartext C&C protocol. Thus observing similarities in sufficient quantity should enable the detection of patterns through artifacts that are derived from the observed similarities in the C&C behaviour. This means that while the knowledge of the details of the cleartext C&C protocol are not necessary for the proposed detection system, the assumption is made that such a protocol indeed exists and is uniform for a single malware variant. This assumption is an implied general precondition for all network-based intrusion detection systems that rely on network signatures, as detection would otherwise not exist. Other types of detection approaches, e.g. anomaly-based systems, can nevertheless still be applied. Therefore the scope of the detection capability presented in this master's thesis covers malware for which the use of a shared cleartext C&C protocol is reflected in the form of side-channel information leaks in the network traffic.

The confined scope and novel approach of the research presented in this thesis constitute the complementary contribution to the state of the art. Malware C&C channels in general have been the subject of extensive research efforts and have resulted in a variety of proposed detection systems. A large part of these approaches focus on C&C infrastructures that communicate over plaintext protocols or expose other exploitable invariant characteristics in the C&C communication. In order to minimize the overlap in the subject with other existing work, further technical boundaries are defined that clearly state the focused nature of this research. In particular, detection efforts are directed at C&C channels that (i) utilize TCP as C&C carrier protocol; and (ii) utilize custom encryption as means to obscure the cleartext C&C communication. Custom encryption refers to encryption routines that don't exhibit an identifyable fingerprint in the carried application-layer protocol, e.g. as is the case for transport layer security (TLS). The belief of the author is that

confining the conceptual scope and introducing well defined technical boundaries to the system allows the creation of a specific and advanced detection.

A similar scope has been explored by Rossow et al. in ProVeX [12]. The approach taken by the authors relies on probabilistic vectorized signatures that are derived from decrypted C&C traffic by evaluating byte probabilities at different positions in messages. Subsequent C&C detection is performed by actively decrypting network traffic and identifying characteristic byte values at specific offsets in the decrypted messages. ProVeX was able to obtain promising results in a live test, where the established IDS system was deployed on a small university network for 24 hours. Due to the limited extent of the test, both in size and duration, it is reasonable to question the scalability of the system. As opposed to ProVeX, the solution presented in this work is applied to actual enterprise environments for extended periods of time and thereby tested for performance, scalability and detection accuracy. The achieved applicability on large-scale enterprise environments therefore sets this research apart from related works, and can as a result be considered a complementary contribution to the state of the art.

3.2 Context and architecture

This section presents the unique context in which the research is realized. This context is comprised of an existing malware analysis platform by Lastline Inc.[3] and the analysis systems utilized by the network analysts in the company. The new detection capability that is acquired as a result of this research adds to the already existing capabilities of the platform, supplements the network analysis framework and improves the detection coverage of the IDS solution. The architectural context is presented in the form of a pipeline in Figure 10. The significance of the steps in the pipeline is explained below by outlining the functions of each phase and describing the intermediate products transferred between the stages. This section provides an overview of the process of generating network fingerprints, which is further described in detail in Chapter 4, where the focus is set on the implementational specifics of the network analysis system.

As mentioned in the previous section, an essential requirement for signature-based intrusion detection systems is the availability of sufficient material, i.e. malware network traffic, for the analysis. In typical research projects this is accomplished by utilizing existing archives of recorded network traffic, and establishing detection against malicious traffic that is blended among a large set of benign traffic in the archives. Another approach to obtain malicious network traffic is to execute malware in a contained execution environment and to record the network traffic generated by the malware within a virtual network. These contained execution environments, often referred to as *sandboxes*, are automated systems for dynamic malware analysis that are utilized to examine and understand the behaviour of unknown malware samples by executing them [44, 45]. These systems have been studied extensively and their detailed functions are out of the scope of this work. However, the presented research

[3] https://www.lastline.com/

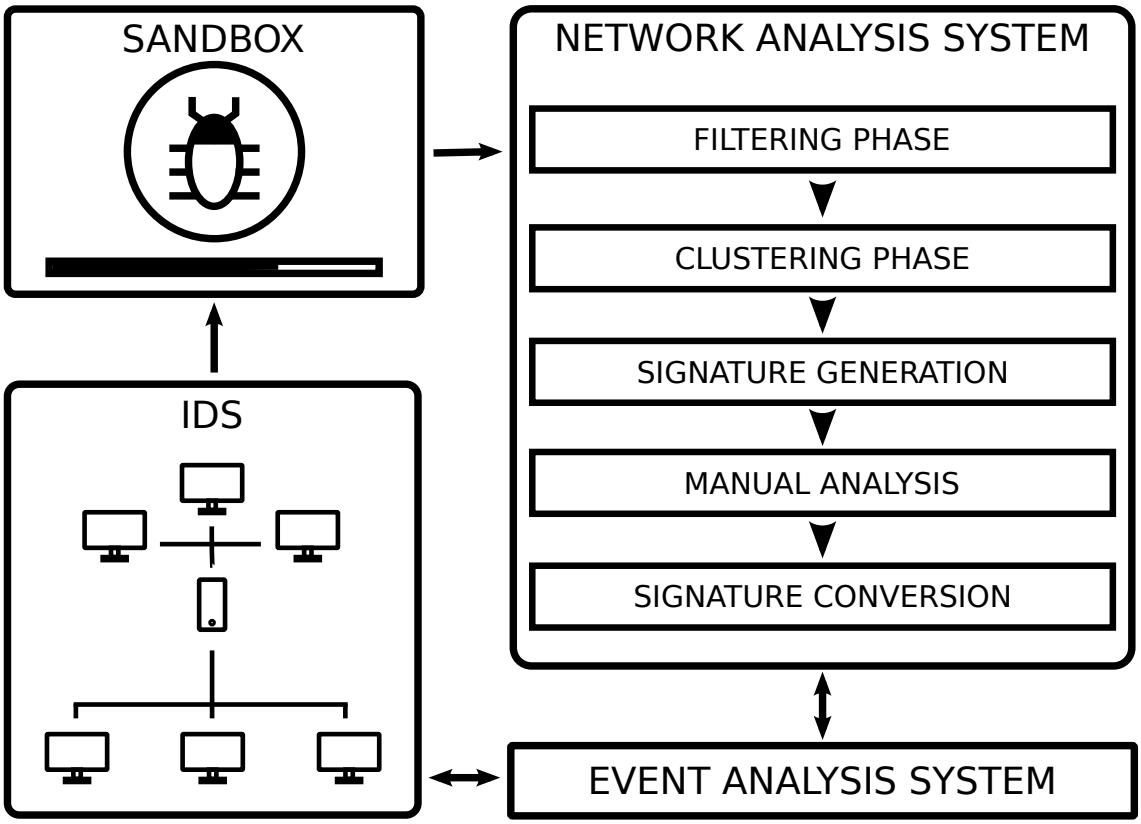


Figure 10: Target analysis pipeline.

utilizes the sandbox provided by the aforementioned malware analysis platform and therefore the general benefits and products of a sandbox require further explanation.

An additional requirement for research projects that utilize sandboxes for generating malicious network traffic are the actual malware samples that are executed and analysed in the contained environment. This malware can be for example collected from existing archives or from public sources, such as VirusTotal [4]. In the context of this thesis, input to the sandbox is generated by extracting transmitted files from real-world production environment networks. Files are extracted by the underlying intrusion detection system, which is positioned at the edge of these networks and monitors the passing traffic. This extraction can be performed on a multitude of different protocols, such as HTTP, FTP, SMB and SMTP. The range of supported protocols in conjunction with a wide adoption of the malware analysis platform offer a substantial stream of input data to the sandbox execution environment. This stream can be further supplemented through manual submission of malware samples. Files that are extracted or submitted manually to the sandbox environment are placed in a queue to go through a thorough dynamic analysis process.

The goal of running a malicious sample in a sandbox is to gather insights of the malware malicious intent on a victim machine. A malware might attempt to evade dynamic analysis by implementing evasion tricks that detect the presence of

[4]https://www.virustotal.com/

a debugger or virtualised environment, and sidetrack the malware execution to a branch that displays benign behaviour. However, performing run-time analysis on the sample with an emulated processor allows a debugger to monitor each instruction the malware executes and to detect code branches in order to divert the execution. This enables to circumvent some evasions and to expose the concealed malicious behaviour, e.g. C&C network activity. Using emulation or virtualisation to simulate a hospitable environment for a malware to run enables the parallelization of sandbox systems. The benefit of this feature is the immense scalability increase that simulated environments provide compared to pure hardware-based dynamic analysis systems. In addition sandboxes can be easily reset to their original state after a malware execution has ended in order to prepare the environment for a new malware analysis.

Sandbox analysis provides therefore a scalable and fast way of collecting malicious network traffic for further analysis. This network traffic, in the form of packet captures (PCAP, or pcap), serves as the input to the next high-level stage in the analysis pipeline, *the network analysis system*, which is displayed in Figure 10 after the sandbox. The streamlined process, where files are extracted from the wire and immediately analysed, offers the advantage of a continuous stream of "fresh" data to the network analysis system. The captured malware samples and their network traffic thus represent a snapshot of the present state of malware in "the wild" and provide a good overview of prevalent threats and ongoing infection campaigns. The C&C traffic in the network analysis system therefore allows for a quick response against new and uncovered malware within a minimal time window.

The network analysis system comprises the core functionalities that form the basis for the developed detection capability of encrypted C&C channels. These functionalities are divided into successive substages where each incorporates actions that analyse and transform the input produced in the previous substage. The initial input to the analysis system consists of the raw network traffic captured by the sandbox environment, and its metadata. The metadata is generated by several application program interface (API) functions that parse the traffic captures and provide flow specific information, such as DNS name, carrier protocol, and source and destination IP addresses. The combined information is stored on the network analysis system in a relational database that allows for efficient data loading.

In order to conform to the technical scope defined in the previous section, network traffic that meets the defined requirements is separated in a preparatory *filtering stage*. This filtering is carried out by utilizing special protocol matchers on the captured network traffic. The protocol matchers are a set of predefined conditions that are checked for each flow in a given traffic capture. These conditions define rules about the structure and form that must be fulfilled by a flow in order to be tagged as belonging to a certain protocol. Thus, in order to comply with the strict scope of this research, all flows that match an application layer protocol matcher's definitions are excluded from the analysis. As a result, all TCP flows that are not matched in the filtering process are included in the analysis dataset. The filtered dataset is then passed on to the next stage.

The second stage in the network analysis system, denoted as the *clustering phase*, forms the most important stage in the generation of network fingerprint signatures.

The goal of this stage is to detect meaningful correlations in the filtered TCP flows and to group intercorrelating flows. This is achieved by clustering the flows and using the individual payload sizes of the captured outgoing requests as the basis for clustering. The clustering stage is the part in the pipeline where the sufficiency of available malware traffic plays a significant role. For clustering algorithms in general, larger clusters indicate either that the boundaries defined for the algorithm are too broad, or that the data in the cluster show good similarity and are highly intercorrelated. Discovering well intercorrelating clusters thus becomes more difficult if the initial data is sparse and doesn't contain a large enough quantity of network traffic generated by an individual malware variant that would represent the malware sufficiently. Hence the importance of a scalable system that produces input data to the network analysis system can't be overstated.

The cluster information is relayed further to the *signature generation phase*. This phase represents the stage in the pipeline where correlations in the network traffic are expressed with a rule language that describes the nature of the correlation in a well defined manner through signatures. The product of the stage are preliminary network signatures that represent network fingerprints. In addition to the TCP payload size information, the signature generation also leverages the side-channel entropy information exposed in the payloads. Even though the entropy information is not included as input to the clustering phase, it is important for improving the precision of network fingerprints. Additional constraints in the signatures lessen the risk of random accidental matches on benign traffic and, if the validity of these constraints is carefully verified, only affect positively the probability of matching on malicous traffic. In conjunction with derived entropy information, the signature generation phase translates the correlation of the payload sizes into a human-readable rule language, which is described in more detail in Chapter 4.

The generated signatures undergo next a manual inspection phase, where the signatures are verified by human network analysts and refined to higher quality. The signatures are then input to a conversion system that transforms them into components of the matching framework that is supported by the employed intrusion detection system. The existing malware analysis platform employs the Suricata intrusion detection system for tapping into passing traffic. Instances of the Suricata IDS are run on physical platforms, called *network sensors*, that are distributed and embedded at the edge of the networks of customers that piggyback on the malware analysis platform to detect threats that possibly infect their computer systems. Network signatures that are deployed on sensors are also called *network detectors*. The Suricata rule matching framework applies the converted network fingerprint detectors to each passing flow that matches the initial requirements of the assumed malicious traffic, and thereby attempts to detect encrypted C&C channels. In a case where the packet payload size and entropy constraints for a defined network fingerprint are found to match a passing flow, the flow data is recorded and transferred along with the detector information to an event analysis system for inspection and verification.

A significant feature in the malware analysis platform is the possibility to deploy network signatures in two different detector modes, *candidate* and *real*. The difference

between the modes is that real mode detectors are forwarded, in addition to the event analysis system, also to a production environment that allows customers to view the events. The benefit of containing candidate mode events in the internal analysis system is the addition of an adjunct verification layer. Events raised by candidate mode detectors are reviewed by network analysts who produce decisions on their validity and verify the quality of the underlying signature. This concept is an integral part of the analysis pipeline in that it offers valuable insights about the behaviour of signatures when they are exposed to large volumes of beningn network traffic. This feature in the platform allows wide scale testing of network signatures before they are deployed in real mode to the sensors and raise visible alerts to customers. Detectors can go through a loop where the underlying signature is edited and verified multiple times before it is deemed to be high-quality. A detector that proves to be efficient performance-wise and quality-wise can then be promoted into a real mode detector.

3.3 Objectives and verification

The main objective of this research work is to establish a proof of concept for the network-based detection capability for encrypted C&C channels. The detection capability is to be free of assumptions about, and unconstrained by, any possible underlying C&C infrastructure that enables the communication of malware, such as centralized or decentralized designs. The established detection capability must also be able to remain performant on IDS systems that tap onto heavily loaded network links that transfer data with speeds up to 1 Gbps. In addition the performance, measured in packet loss (%) and processing load, must remain within reasonable boundaries with respect to the added benefits of new detections. These boundaries are determined in the experimental stage of the research when the qualitative attributes of the detection become more apparent.

From the state of the art's point of view, the goal is to produce a novel way of perfoming C&C channel detection with network fingerprints. This goal is evaluated through a review process on the generated events in the event analysis system. The qualitative success and reliability of the network fingerprints can be measured e.g. as the ratio of generated true positive (TP) events with respect to generated false positive events. This approach, however, can not be viewed as an exclusive measure due to the analysis loop enabled by the event analysis system, which allows a continuous improvement and verification process of non-performant network fingerprints. Thus the success of network fingerprints could be also expressed in a quantitative manner, e.g. as number of events that would go otherwise undetected. However, due to the use of real-world networks as a testing environment the tracking of missed detection events, i.e. false negatives (FN), is not possible in the context of this thesis.

These main objectives necessitate the accomplishment of several subgoals. Network fingerprints are enabled by the analysis framework and the signature generation process. The establishment of the detection capability requires constructing new components that provide the necessary tools for analysing incoming traffic and extracting interesting features from flows that meet the criteria of the research.

These components must be compatible with existing network analysis systems and allow the creation, verification and deployment of network fingerprints in an efficient and reliable manner. These subgoals can only be verified in a subjective manner as quantifying the usefulness of an analysis system that is dependent on human involvement is difficult.

3.4 Summary

Command and Control traffic has become more complex and increasingly non-trivial to detect. The current state of the art focuses on invariant-based detection mechanisms that are thwarted by encrypted C&C channels. When implemented on top of non-descriptive carrier protocols, these C&C channels don't exhibit any invarints to be used as basis for conventional pattern-based detection. With changing encryption keys these channels implement a type of polymorphism in their communication, as each channel conveys dissimilar messages, thus erasing any payload intercorrelation of two separate malware instances. These C&C channels can, however, be targeted assuming that the communication exposes side-channel vulnerabilities. This thesis exploits this idea and introduces a new detection technique, network fingerprints, that takes advantage of side-channel leaks in order to extract fingerint signatures that accurately identify encrypted C&C channels. These network fingerprints target reoccurring C&C protocol characteristics of malware and leverage the payload size and entropy information of captured packet sequences for detection. This detection capability is enabled through supplementing an existing network analysis framework with additional components that are responsible for the analysis of the traffic data and signature generation. These network fingerprints are then deployed to real-world production environment networks where they are exposed to large volumes of benign traffic. The performance of the detection capability is evaluated quantitatively and qualitatively by inspecting performance metrics on network sensors and by deriving information about true positive and false positive events.

4 Experimental setup

This chapter presents the individual phases that are constructed or supplemented to the network analysis infrastructure in order to establish a proof of concept system for the network fingerprint-based detection. The chapter clarifies and explains design choices and technical details about the implementation of the separate components. The first section describes the process in which the non-descriptive TCP network traffic is filtered from the large stream of traffic captures provided by the sandbox analysis system. The filtered traffic is input to the clustering phase, which groups intercorrelating C&C flows in preparation for the signature generation phase. This is described in the second section. The third section presents the heuristic method that extracts entropy information from the generated clusters and feeds the combined information to an automatic signature generation algorithm. The results of the signature generation phase are visualized graphically in the next phase, which is described in the fourth section. In this phase the signatures are verified and refined by network analysts to obtain the most accurate and descriptive network signatures. The fifth section explains the structure and components of the final form of network fingerprints and how traffic matching is performed in an IDS system. The last section provides a short summary of the whole chapter.

4.1 Protocol filtering

Network traffic captured in the sandbox system has to be filtered before analysis can be applied to the traffic data. The introduced scope of this research states that the focus is set only on bare TCP traffic and thus all application-layer protocols, and other transport-layer protocols, are by definition not part of the research topic. The protocol filtering is done to assure an accurate and performant outcome for the new detection technique rather than providing an all-round solution that is not capable of performing precise detection. Restricting the set of targeted network traffic allows the analysis to focus on specific features of the exchanged packets and to extract accurate fingerprints to describe the C&C communication.

Protocol filtering is done by comparing a wide range of known protocol implementations to the data transmitted in a network flow. A common property of application-layer protocols is that they conform to well-known definitions of protocol structures, which facilitates the identification of the transport-layer protocol's payload. A standard practice for recognizing the contents of the transmitted payload is to loop over a list of protocol parsers and to scan packet payloads for structures. A protocol parser specifies a set of conditions that define the boundaries for the payload in order for it to match the parsed protocol. These boundaries are derived from public specifications, e.g. Request for Comments (RFC), or other public sources that clearly define how a protocol is to be implemented. However, due to the frequent phenomenon of bad protocol implementations, protocol parsers must compensate by

```
1  PASS hax0r
2  KCIK n{US|XPa}tiwuwqy
3  RSSR tiwuwqy 0 0 :tiwuwqy
```

Listing 1: Mangled IRC commands from the Dorkbot malware.

introducing tolerance to the parsing process.

In this work, protocol parsing is performed by an API that is implemented in the Python programming language[5]. The boundaries for a protocol are defined with regular expressions that are applied to the first packet for each flow in the input set of traffic captures supplied by the sandbox environments. A protocol parser can comprise multiple regular expressions conditions that each define varying constraints for the payload and at the same time constitute tolerance to the matching. If the first packet of a flow matches a regular expression condition of a protocol parser, the flow is tagged to be carrying traffic from the parser in question. Flows that don't match a protocol parser's boundaries are matched against all other available protocol parsers until a match is found or until all parsers have been exhausted. Some protocols, e.g. messaging protocols that implement command-based communication, can be assigned for more stringent inspection, where each packet is verified separately and the flow is tagged only if all packets can be matched. If a flow doesn't get tagged by any protocol parser, it is interpreted to not carry an application-layer protocol and is therefore marked as an unknown TCP flow. From the scope of this work, and the implementation of these protocol parsers, follows that these unknown TCP flows, which are left unmatched in the protocol filtering process, form the dataset that is studied.

Allowing tolerances for protocol parsers helps in identifying the correct application-layer protocol of a flow and facilitates the separation of malicious and benign traffic. Some malware employ modified benign protocols that have been altered to suit the needs of the attacker. This behaviour is shown for example by Dorkbot, which is a Trojan horse malware that spreads through social media messages and is used to steal personal information, such as banking credentials [46]. Listing 1 shows three messages that constitute an actual initial check-in request made by the bot, where the proper IRC commands `NICK` and `USER` have been mangled to `KCIK` and `RSSR` [47]. These instances are easy to discard with protocol parsers that perform deep inspection on all packets.

Although these protocol parsers pose an effective way of separating known protocols from unknown traffic, the process includes inevitable shortcomings. Firstly, it is acknowledged that it is impossible to be aware of the existence and specifics of all application-layer protocols. There exist uncommon and not widely adopted protocols that are linked to benign services and used by benign applications. Missing protocol parser implementations can cause these protocols to remain untagged in the matching process. Secondly, mangled or otherwise incorrect protocol implementations that don't parse correctly receive similar treatment. Due to these effects, the pool of flows that is passed next to the clustering phase can contain unquantifyable amount of

[5]https://www.python.org/

noise, which can have detrimental effects on the clustering results.

4.2 Clustering unknown network traffic

The clustering phase concentrates in finding correlations in the filtered network traffic. The clustering is performed on feature vectors that are extracted from individual flows in the traffic dataset. The features that form the grounds for the clustering are payload sizes of sequential client request transactions in an established TCP flow. Server transactions are not incorporated into the clustering phase due to the simple facts that (i) C&C module implementations of malware are often static and subjects of infrequent updates; and (ii) client-side malware often repeat messages in their communication. Recurring messages can be caused for example by repeating attempts of forming a connection to an unreachable C&C infrastructure, or by design in the form of periodically sent messages, often called *keepalive* messages, whose function is to keep the connection to an C&C entity open for an extended period of time. Furthermore, malware may implement C&C protocols that operate with the premise that the receiving C&C entity remains silent and doesn't respond to a bots routine messages, e.g. check-ins and data uploads, unless new commands are to be issued. Concentrating only on client originated transaction payloads therefore offers a more probable way of discovering intercorrelations in the dataset.

The sequence of payload sizes that forms the feature vector for a flow is initiated from the first request of the flow. This convention is based on the assumption that initial messages exchanged over C&C channels characterize a malware C&C protocol better than messages that appear later in the C&C conversation. Initial messages of a bot, e.g. check-ins and configuration information uploads, set the premise for the subsequent instructions issued by the controlling C&C entity. This premise may include a setting where the bot doesn't engage in recurring C&C communication that would provide an alternative sequence of correlating messages. Initial messages thus represent a set of requests that are more likely to show similarities for a malware variant and are therefore more significant for the clustering.

The clustering algorithm utilized in this research project is Density-Based Spatial Clustering of Applications with Noise, more commonly known as DBSCAN [48]. DBSCAN is an algorithm that performs clustering according to a density-based connectivity analysis. The algorithm attempts to find *core points*, i.e. points that are densely surrounded by other points, and connect these to create dense regions as clusters. The parameter ϵ (eps) defines the radius of a circle around a point for which the area contained by the circle is called the point's ϵ-neighbourhood. The *density* of a point's ϵ-neighbourhood is measured as the total number of points that are within ϵ-radius of that point. The parameter *minPts* specifies the minimum number of points that have to be in the ϵ-neighbourhood of a point for the neighbourhood to be considered *dense*, and for the center point to be considered a core point. A cluster is defined to be the set of points in the dense ϵ-neighbourhood of a core point. In this thesis the distances between points are expressed with the Euclidean distance. [48, 49]

Given a dataset, a point p in the ϵ-neighbourhood of a point q is said to be *directly density-reachable* from q if q is a core point. This notion of direct reachability is not symmetric, i.e. p being directly density-reachable from q doesn't imply that q is directly density-reachable from p. However, reachability is chainable in a non-direct manner, i.e. a point r can be said to be density-reachable from point q if point r is directly density-reachable from point p, and moreover point p is directly density-reachable from point q. Furthermore, any points p and q are said to be *density-connected* if both are density-reachable from a point o. From these definitions follows that all points in a cluster are mutually density-connected, and any point in a dataset that is density-reachable from a cluster is part of that cluster as well. [48, 49]

Given the parameters, ϵ and minPts, DBSCAN propagates by looping over all points in a dataset. For each point the algorithm performs a region query that determines whether the ϵ-neighbourhood of the point contains at least minPts points. If the minPts condition is not realized, the point is labeled as noise. If the condition is realized, the point is labeled a core point and all points in its ϵ-neighbourhood are added to a candidate set for which the process is performed next. The repeated manner of processing a core point's neighbours allows the cluster to be expanded and to comprise multiple core points. When all points in the candidate set are processed, the algorithm continues by selecting another point from the dataset that has not been visited yet. [48, 49]

DBSCAN offers a simple way of clustering network traffic flows. Depending on only two initial parameters, ϵ and minPts, DBSCAN is able to perform clustering on datasets of arbitrary shape. This feature is desireable, as the nature of malware network communication is difficult to model. Enabled by the property of chainable reachability, the flow feature vectors can contain deviations that cause small scattering without affecting the result of the clustering. Given the premise of this research, changes in the payload sizes of a single malware variant can be attributed to the changes in the payload contents. Thus big deviations can indicate different types of transmissions, which are acceptable to be assigned to separate clusters. Parameter estimation and expert knowledge on the underlying dataset allow adjusting the parameters in a manner that produces the best results.

For the purpose of this thesis a modified version of DBSCAN was implemented, which is presented in Appendix A. The main difference to the pseudocode provided in [48] is the inclusion of an additional filtering procedure after each region query. If a region query for a point fulfills the minPts requirement, the created set of candidate points is scanned for identical points and all indenticals are labeled to belong to the cluster without explicitly visiting each point. This decision is necessitated due to some malware, e.g. worms, that produce large quantities of identical requests that can cause the execution of the algorithm to slow down significantly. Another noteworthy detail in the custom implementation is the exploitation of the property in DBSCAN that enables the usage of an arbitrary index structure for the region queries. Because the region queries don't assume any particular indexing method, average computational complexities of $\mathcal{O}(n \log n)$, where n is the size of the datset, can be achieved by using accelerating index structures. This is a particularly great benefit when clustering large datasets, like in the case of this work. The utilized

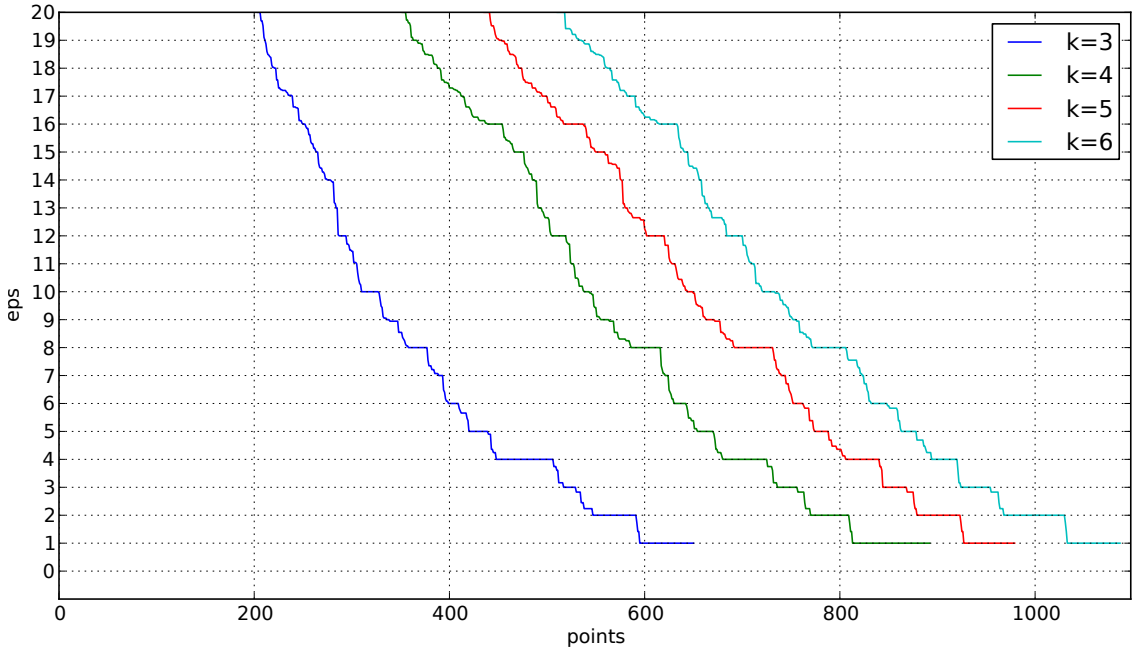


Figure 11: k-NN estimation of ϵ for three-dimensional feature vectors (07.01.2016).

DBSCAN code that is provided in Appendix A takes advantage of this by utilizing an implementation of the KDTree spatial index structure from the scikit-learn[6] Python toolset.

In order to obtain good results from the clustering algorithm, the parameters ϵ and minPts need to be optimised. Ester et al. [48] describe an approach that uses the k-nearest neighbours, or k-NN, method for estimating the value of ϵ. In this method, for each point in the dataset, the k nearest neighbours are searched and the distance to the most distant neighbour, i.e. the k-th neighbour, is recorded. The obtained set of point-to-distance mappings is sorted in descending order based on the distance and plotted in a *sorted k-dist graph*, where the y-axis shows the distance values and x-axis the points. This graph illustrates for a given k, which denotes a choice for minPts, the set of points that would be part of a cluster for a choice of ϵ. According to Ester et al., an optimal threshold value can be found at the first point in the first "valley" of the sorted k-dist graph. All points that are mapped to a higher k-th distance value would be labeled as noise during the clustering, whereas all points that have lower values would then naturally be part of a cluster. [48]

Training an algorithm to automatically detect an optimal value for ϵ is difficult, but it is relatively simple for a human to estimate the threshold value in a graphical representation. Therefore the approach of visualizing the k-NN estimation is also taken in this research work. The results of running the algorithm are shown in Figure 11, which portrays four k-dist graphs for different values of k. The dataset used for this estimation was extracted from the database on the 7th of January 2016 and contained all unknown TCP flows within a window of 30 days. The dataset was

[6] http://scikit-learn.org/

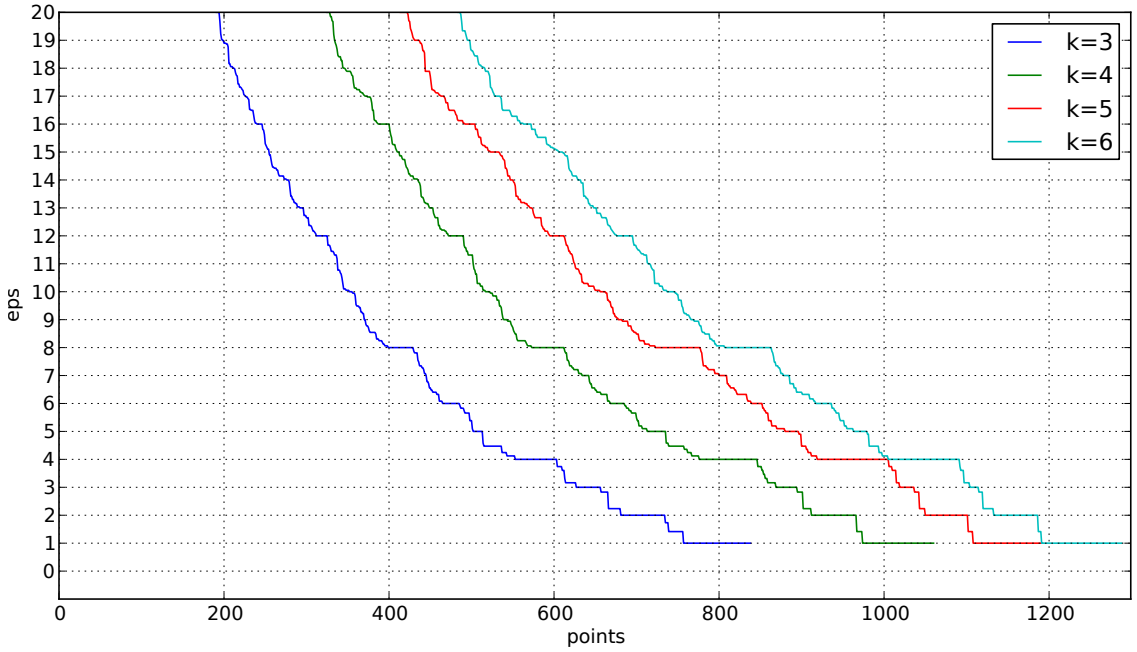


Figure 12: k-NN estimation of ϵ for three-dimensional feature vectors (07.02.2016).

further filtered to only contain flows that have a client request sequence that contains at least three transactions. This means that the feature vectors were comprised of three request payload sizes. In the figure, the effect of increasing k, i.e. minPts, can be seen as a right shift on the x-axis. This shift can be explained by the inversely proportional relationship between the number of obtained clusters and minPts, where a higher constraint on the minimum cluster size naturally leads to a smaller number of obtained clusters.

The graphs in Figure 11 show two good candidates for a threshold value for ϵ. These are $\epsilon = 4$ and $\epsilon = 8$. The reasoning for this is that for all values of k both of these ϵ values show a small bend in the graphs. These bends are emphasized by the straight horizontal lines, which represent situations where multiple values have the same k-th distance to a neighbor. Therefore a straight line indicates that for the corresponding ϵ value on the y-axis a relatively large set of points may be included in the obtained clusters thereby offering a fitting choice for the threshold value. The choice between a higher or lower threshold point depends on the desired properties of the output. A higher ϵ value will produce a larger number of clusters, but features the possible drawback where the cluster quality may be affected negatively through the inclusion of noise, i.e. points that would be outliers with a smaller ϵ value. A lower ϵ value in contrast will produce less noisy clusters, but in a smaller quantity.

As with the DBSCAN algorithm itself, identical points can cause problems to the estimation of the threshold value. For a single feature vector, for which the number of identical instances in the dataset is equal or larger than k, the k-th distance is 0. Because the minPts condition is satisfied with k identical points, the k-th distance calculation doesn't consider other points that could be theoretically scattered densely in the region of the point. Including identical values in the k-th distance calculation

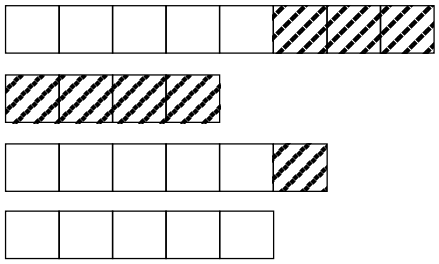


Figure 13: Example of the prefiltering of feature vector dimensions with $D = 5$.

therefore results in an inaccurate metric and provides a misleading presentation of clusters. On the other hand filtering identical points completely from the estimation process would generally result in too high k-th distance measures, as points, which would fall outside of the ϵ-neighborhood of a point during the clustering, would be included in the k-th distance estimation. To avoid the side-effects of completely including or excluding indentical points, the estimation process jumps over the calculation for points that have an equal or larger number of identicals compared to k. This has the effect that for all values of k, the number of points included in clusters for a value ϵ is higher in reality than what is shown on the graph.

Continuous analysis on the data provided by the sandbox environment has shown that the results of a k-NN estimation for ϵ remain fairly constant over time. This is illustrated in Figure 12 where a k-NN estimation was performed one month after the previously presented estimation using identical parameters. Although the dataset loaded from the database is completely different, the threshold values $\epsilon = 4$ and $\epsilon = 8$ still remain good estimates for the new dataset. This could indicate that the malware network traffic in the datasets is highly similar between the two 30-day windows. However, the number of flows included in the window has increased by approximately 200.

Unlike ϵ, the value for minPts is not estimated algorithmically in this research, but is instead chosen manually based on knowledge and expertise on the dataset. In the original DBSCAN paper, Ester et al. eliminate the problem by simply stating that $k = 4$ is a good estimate for all two-dimensional datasets [48]. This value, however, is not suitable for the purpose of this research, as the scope extends to higher dimensions. Therefore the value for the mimimum number of points that can form a cluster must be reasoned based on evaluating the possible side-effects of different choices. A too high value for minPts can result in undesired exclusion of interesting, but small, density regions from analysis. A too low value, on the other, hand may cause effects of undersampling that can lead to lower quality signatures. For the purpose of generating network fingerprints for specific C&C threats, the confidence in the quality of the clusters must be high, which increases the value constraint of minPts. Experiments and manual analysis carried out during the realization of this proof of concept system have shown that minPts values from 3–6 are necessary to gain deep understanding of the dataset. To achieve this, the DBSCAN algorithm is run multiple times for different values of minPts and the results are stored separately for subsequent analysis.

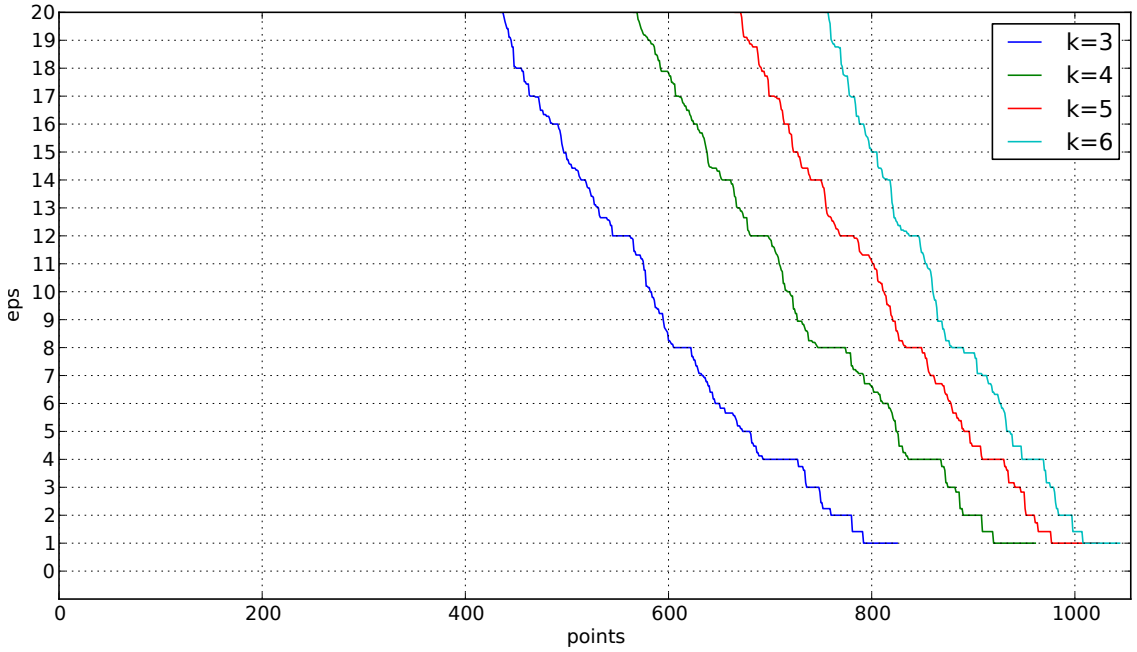


Figure 14: k-NN estimation of ϵ for five-dimensional feature vectors (07.02.2016).

In addition to ϵ and minPts, the choice of the number of dimensions for the clustering phase has significant implications for the clustering output and the whole analysis pipeline. Dimensions refer here to the number of client request payload sizes contained in the feature vectors. Before clustering, the input dataset for DBSCAN is prefiltered to contain only feature vectors with D or more features, where D denotes the dimension. Specifically, feature vectors that contain less than D features are dropped completely from the dataset, and feature vectors that contain more than D features are cropped to the length of D. This is illustrated with a simple example in Figure 13, where the prefiltering is performed on 4 feature vectors when $D = 5$. Striped areas in the figure depict discarded or cropped feature vectors.

As a result of cropping and dropping feature vectors from the dataset, the choice of D affects the number of generated clusters that are available for the signature generation. In addition, D also directly affects the reliability and confidence of a network fingerprint signature. This is due to the fact that the reliability of network fingerprints is dependent on the quantity of client request constraints, as already mentioned in 3.1, and therefore longer feature vectors in practice provide a premise for higher-quality signatures. However, while inducing stronger signatures, a high dimension delivers less signatures due to the dropping of short feature vectors. The effect of dropping feature vectors on the number of available points for clustering can be seen by comparing Figure 12 with Figure 14. In Figure 14 the same k-NN estimation was performed as in Figure 12, but with $D = 5$. As a result, the overall number of available points is reduced by more than 200 for each k. Because a low dimension in turn can induce weaker signatures, finding the right balance in signature quality and quantity through D can be difficult and depends on the wanted properties of the outcome of the clustering phase.

After estimating parameters and prefiltering the dataset dimensions, the DBSCAN clustering algorithm can be run. The output of DBSCAN is labeling information about the generated clusters for the dataset. The label that is assigned for each point describes to which cluster a point is assigned, or that the point is an outlier. The goodness of the clustering and the validity of the labels is not separately verified after running DBSCAN. Clustering in general can be viewed as an unsupervised learning task and analysing the validity of generated clusters is intrinsically hard [19]. This is further complicated by the difficulty of creating truth labels for malware C&C traffic, which can take arbitrary forms and contain diverse communication patterns. Due to these reasons the clusters are validated only later in the analysis pipeline, when the automatically generated signatures undergo a manual inspection and refinement process. As a result, the labeling information is passed without further processing to the signature generation phase

4.3 Automatic signature generation

The signature generation phase concentrates in the automatic generation of network signatures based on the cluster information produced in the previous phase. In this process, payload size information of a cluster is combined with aggregated statistical metrics of the individual flow payloads in order to create a precise fingerprint description of the cluster. In particular, the produced network signatures leverage the Shannon entropy of the indidual paylods that function as a secondary measure with the goal of improving the reliability of the signatures. The signatures output in this phase represent the final network fingerprints and serve as the basis for the manual analysis in the next phase.

The signature generation algorithm loops over each cluster by filtering the initial dataset using cluster labels as keys. The algorithm thus has access to the raw payloads of the flows that belong to a cluster, and can perform computations on the payloads in order to derive other metrics in addition to the payload sizes. The output of the algorithm for one cluster is a single network signature that follows the definition of network fingerprints from Section 3.1. The payload data for a cluster is transformed into tuples of a signature by looping over the payload sequences in an index-based manner and subjecting all individual payloads at an index i in the payload sequences, where $i \in [1, D]$, to the processing at the same time.

In the context of this thesis the metrics that are computed for the payloads at an index i are the payload size and Shannon entropy. A payload size constraint constitutes a stronger constraint to the payload matching than entropy, and is used as the main matching feature for the targeted encrypted C&C channels. This is due to the fact that certain protocol behaviour, e.g. handshakes, are often directly reflected in the sequence of payload sizes. Entropy on the other hand relays information about the format and possible encoding of the traffic. The Shannon entropy of English text is approximately 2.8 bits per byte, while for a totally random sequence of bytes it is 8 [43]. Encrypted communication often reaches entropy values close to 8 and can therefore be identified from cleartext traffic. A high entropy, however, doesn't alone

guarantee that a payload is encrypted: legitimate compressed data, e.g. images, audio and videos, typically show high entropies as well. Therefore entropies function as a reliability and precision improving factor in the signatures on the side of payload size constraints.

The values inserted into a tuple depend on the distributions of the computed metrics. Due to the explained confidence difference that a payload size constraint and an entropy constraint exhibit on the signature, the handling for the two metrics is divergent. A payload size constraint is always entered into the tuple regardless of the distribution of the values, because the clustering, assuming sensible parameters, provides guarantees that the range of values will not be arbitrarily spread. An entropy constraint, on the other hand, is included only if the standard deviation of the values is small. This is because the entropy value distribution is not constrained in the process and can show high variability.

The Python pseudocode for this step is presented in Appendix B. This example illustrates the decisions and values that are inserted into the tuples that constitute a signature. The payload sizes and entropy values are precomputed and stored in the two lists `payload_sizes` and `entropy_values` that are of size $N \times D$, where N is the number of points in the cluster and D is the dimension. The decision on the payload size is straightforward: if all payloads at an index i have the same size, then that value is used; if the values are distributed, then the values form a range constraint, where the range is expanded at both ends by 1 to allow small deviations. In the code, a range is denoted by `X<>Y`, where the X is the minimum value of the range and Y is the maximum value.

As mentioned before, entropy values undergo a different handling. In the case where all entropies for an index i are the same, the entropy is entered into the tuple as a range, where both ends are expanded by a defined margin value, `ENTROPY_MARGIN`. If the values are distributed, they are only useful if the distribution is dense. Therefore if a condition for a maximum standard deviation, `STDEVIATION_THRESHOLD`, is not met then no entropy value will be inserted into the tuple. If the condition is met and the mean entropy is above a defined threshold, `ENTROPY_THRESHOLD`, then only the minimum value of the range, subtracted by the standard deviation of the values, will be used as a constratint. In the other case the mean entropy is low and the constraint is inserted as a range, where both ends are expanded by the standard deviation.

When all D indexes of a cluster have been processed, the tuples are translated into a simple human-readable signature format. This format provides a clear presentation of the conditions of a signature and is used to describe complex queries that are made to the malware network traffic database that is present in the network analysis framework. Each tuple is translated into a segment, which defines keywords about the directionality of the tuple, its index and the payload size and entropy constraint. An example of a translated tuple could be `to_server; client.idx=1; dsize=320<>323; entropy=550<>590;`, which targets the first client request that has a payload size between 320 and 323 bytes, and an entropy between 5.5 and 5.9. Noteworthy are the 1-based indexing and the multiplication of the entropy values by 100 to avoid floating point number operations. These segments can be chained to establish a description for a complete signature. Further details about the translation and the signature

format are outside of the scope of this work.

The generation process attempts to translate the essential side-channel information that was captured in the clustering phase and present it in a flexible format. The generated signatures represent the final network fingerprints and are used as a basis for the analysis in the next phase. Even though this thesis focuses in the extraction and translation of payload sizes and entropies, the framework supports additional metrics that can be leveraged in future work.

4.4 Graphical signature and cluster analysis

The automatically generated signatures undergo next a manual analysis phase. The main goal of this phase is to produce refined and high-quality signatures for the subsequent conversion process that transforms the signatures to a format that is deployable to the Suricata IDS. To achieve the goal, multiple steps are taken in order to ensure the validity and reliability of the signatures, and the maliciousness of the traffic that is matched by the signatures. The process involves collecting and combining information from multiple sources and visualizing gathered data in an illustrative manner in a versatile and effective analysis system.

The importance of human involvement in network traffic analysis can't be overstated. Malware authors constantly adapt their techniques and introduce new ways to evade detection in response to disruption efforts by security professionals. To achieve efficiency and scalability, new detection models often employ machine learning algorithms and utilize automatic signature generation. Applying these models on real networks requires deep knowledge of the domain, and periodic tuning of the algorithms. Therefore, gaining thorough understanding of evasion techniques and common C&C traffic patterns is an integral requirement for enabling precise detection. Understanding the models and threat domain are thus significant factors in evaluating these models and verifying the validity of the traffic that is matched by the applied methods.

In this work manual analysis is employed for the purpose of validating the results of the clustering and automatic signature generation, and refining their quality. A notable side-effect of the protocol filtering is that legitimate network traffic that employs custom or otherwise uncommon application layer protocols may potentially pass through. This effect is enabled by the implementation of the matching process, where all unmatched TCP traffic is assigned to the dataset that is input to the clustering phase. The quantity and consequental effects of such incidents for the subsequent analysis phases are mitigated through the notion of outliers in DBSCAN, which allows the identification and discarding of uncorrelating network traffic. Nevertheless, some benign network traffic might pass through both these phases and end up in the queue for signature generation. Therefore, enabled by the clear and simple human-readable format of the generated signatures, the manual analysis phase serves an important purpose for the quality assurance of the detection capability.

To facilitate manual analysis, a visualization tool was built to illustrate products

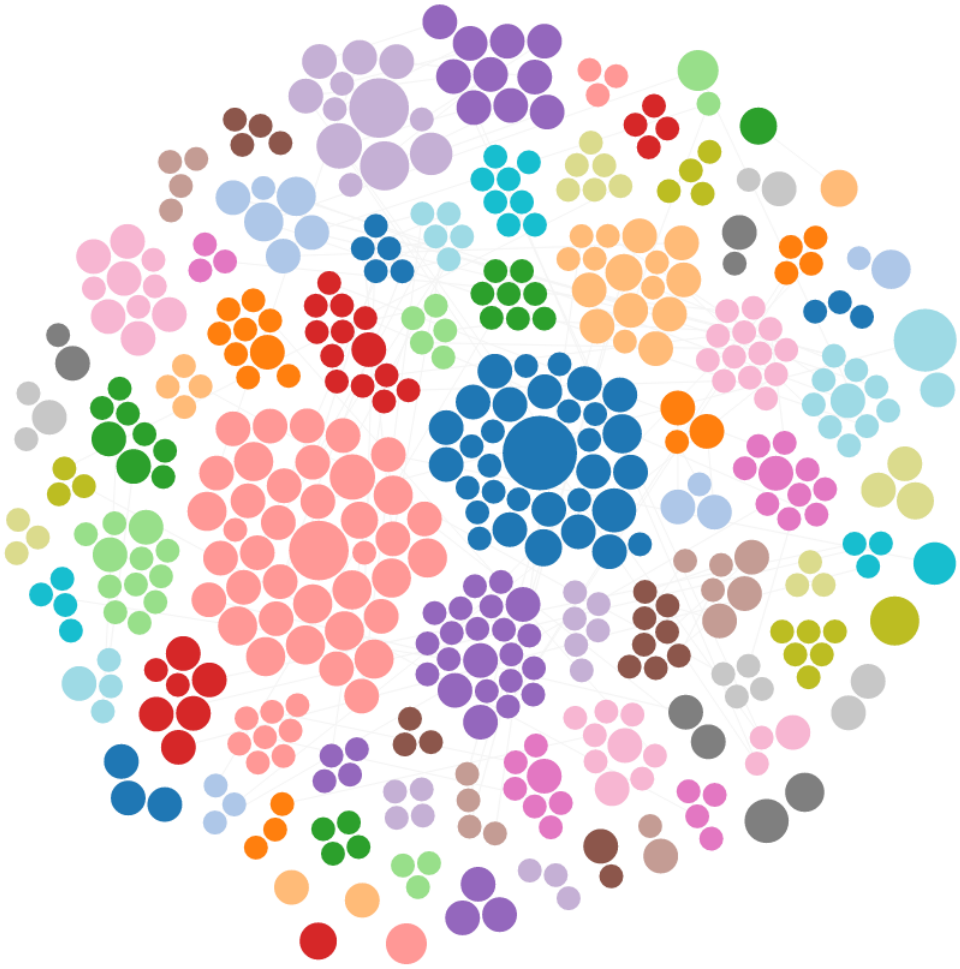


Figure 15: Graphical cluster and signature presentation.

of the previous phases. This visualization tool, created with D3.js[7], JavaScript and HTML, is depicted in Figure 15, where 80 signature clusters and their interrelationships are presented. In the figure, each cluster corresponds to a signature and is depicted as a coloured tightly packed group of circles. Each circle in a cluster represents a single malware sample that was executed in a sandbox environment. The radius of a circle reflects the number of flows that were generated during the sandbox analysis and that are matched by a signature. Cases, where a sample has generated flows during the sandbox execution that match on different signatures, are illustrated with faint grey lines that connect two circles of the same sample in different clusters.

The visualization tool provides a comprehensive overview of the signature generation products and provides helpful information to an analyst about the clusters, samples and flows. The number of circles that form a cluster relays information about the corresponding signature. A large cluster can indicate for example either that a disproportionate number of samples from a specific malware is present in the

[7]https://d3js.org/

dataset, or that the generated signature describes a sequence of requests in a manner that is too relaxed and therefore matches samples from several malware. Insight to this problem can be gathered from the sizes of the circles in the cluster. In particular helpful are the facts that malware samples of single variant typically exhibit similar network behaviour and that the number of matching flows for a sample is reflected in the size of the circle. Therefore clusters with uniform distribution of circle sizes can indicate that the samples in the cluster are from the same malware. If, on the other hand, the size distribution is clearly not uniform, the signature of the cluster is likely to match on multiple malware.

The effect of the choice for the clustering parameters becomes particularly apparent with small clusters. Due to the fact that the clustering is performed on single flows instead of single samples, a cluster can be solely comprised of flows generated by a single sample. Examples of this can be seen in Figure 15 closer to the edge of the visualization. Small clusters in general are an important target of analysis, as uncommon malware, or malware that are used in isolated and targeted attacks, would presumably produce only a limited presence in the dataset. The reliability of an accurate signature constituted by a small cluster is however less reliable, because its generation is based on limited sampling of the malware. Attempts to improve the reliability can be made by tuning the ϵ and minPts values of DBSCAN. The effects of changing these parameters may have particularly strong effects for small clusters, as for example increasing minPts, or lowering ϵ, may result in the exclusion of a small cluster from a clustering run. Experimenting on the input parameters is therefore useful for uncovering important clusters.

To aid an analyst in the analysis process, the visualization tool provides information about the displayed samples in an interactive manner. This is implemented by tracking mouse click events and showing related information on the target of the selection. The interactive user interface (UI) is depicted in Figure 16. For example, connections between circles can be highlighted by clicking on a circle. This has the effect that linked circles are selected, while the rest of the circles are filled with a grey colour. Furthermore, double clicking a circle expands the selection to the current cluster and any circle that is linked to a circle in the cluster, which can also be seen in the figure. Selection events on circles invoke an API call that loads useful threat intelligence from a database. This information is displayed on the left side of the UI and includes details about the selected malware samples, the threat class and family of the sample, the analysis score in the sandbox and the detection ratio in the VirusTotal malware scanning service. In addition, information is shown about the currently selected circle and all circles that are linked to the selection. The tool currently doesn't implement the functionality to map a signature to the visualization, e.g. through a search function, but this is part of planned improvements.

The information that is displayed in the visualization UI is used as a basis for decisions about the quality of the signatures. A signature can be defined to be of good quality when at least the following points are fulfilled: (i) the signature matches the communication of only a single specific threat; and (ii) the signature coverage extends to all communication of a specific sort for a threat. The first condition states that signatures should not cover C&C traffic of multiple malware. Constraining

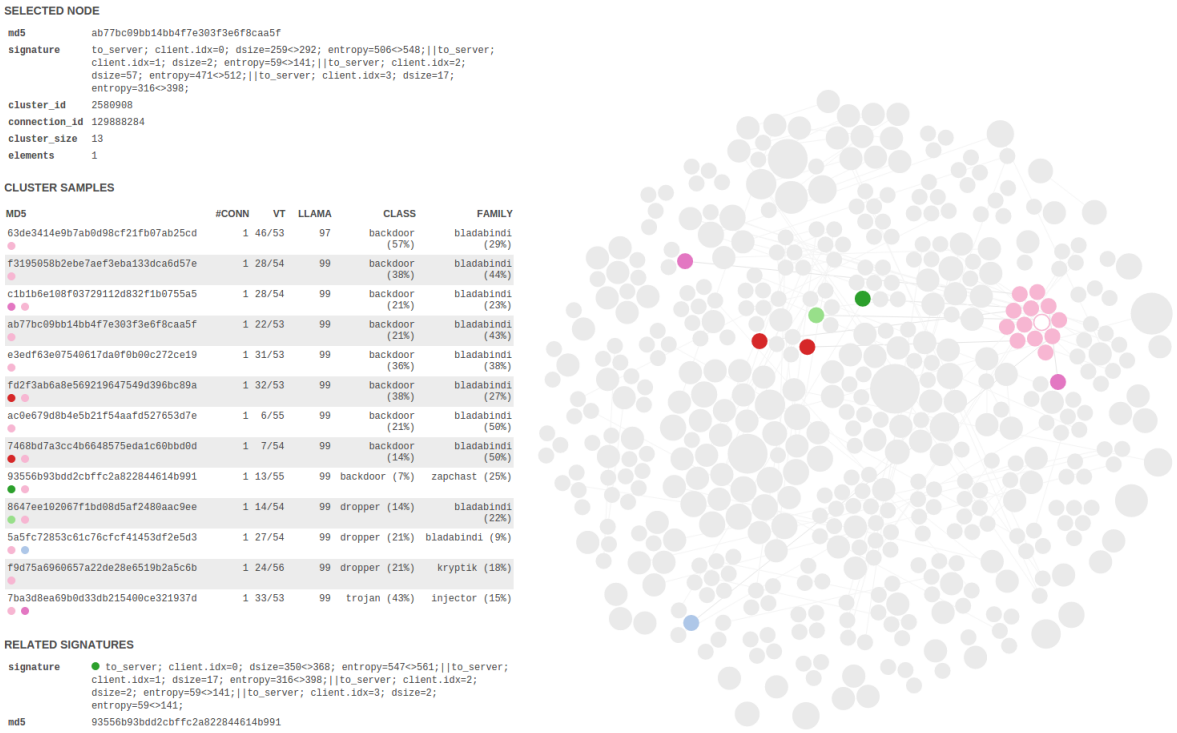


Figure 16: Performing graphical cluster and signature analysis.

signatures to cover only a single threat allows for accurate threat attribution and decreases the risk of defining too broad signatures. The second condition states that in an optimal case a signature should cover well a type of C&C communication that can contain variants. For example, this means that a check-in of a malware should be matched also in cases where the check-in contains variants. Due to the limited material and information that is available for analysis, these conditions should be realized to the extent that is possible.

The analysis process for a signature takes into account multiple aspects when producing a decision. A major aspect in the signature analysis is the composition of the clusters, which is tested for compliance with the first condition that was mentioned above. The threat information in the UI is aggregated data that has been loaded from various sources, most notably internal threat intelligence storages of the existing malware analysis platform and VirusTotal. The aggregation procedure that is run in the network analysis system attempts to derive meaningful threat classes and families for each sample that has been analysed in the sandbox environment. An example of this data can be seen in Figure 16, where the threat information for the samples of the selected cluster is displayed in a table. A potentially good signature is characterized by a unique threat class and threat family. This scenario is, however, not always fulfilled, because threat attribution is difficult and ambiguous across anti-virus vendors' results in VirusTotal, and can include inaccuracies [50]. These cases require deeper analysis on the network traffic generated by the samples in order to verify whether they belong to the same threat class and family. This scenario is also portrayed in the figure, where the samples would undergo a further

manual analysis to verify whether the backdoor "Bladabindi" is an accurate threat family for all of the samples.

Another major aspect in the analysis process is the general verification of maliciousness of the samples in a cluser. This verification is a standard process for all clusters, but is necessitated in particular by clusters that don't have aggregated threat information available. An example of a possible cause for this scenario are novel malware that haven't been subject of widespread analysis and naming yet. A general maliciousness verification can be based on a number of information sources, but due to the availability of an advanced sandbox environment the decision is largely based on the results of the dynamic sandbox analysis.

The available sandbox employs full system emulation (FUSE) to analyse samples. This enables a full instruction by instruction view into the execution of a sample and provides accurate detection and fingerprinting capabilities of the runtime environment. The analysis system can thus detect and circumvent attempts by the malware to identify and evade the dynamic execution. As a result of the resistance to detection, in an optimal case the sandbox is able to induce the same malicious behaviour from a malware sample that it would exhibit on a normal infected machine, including network traffic. Due to the high sophistication of the sandbox environment, the produced analysis report can be used as a reliable basis for the decision of a samples maliciousness. This decision is, however, always supplemented with information from other sources, such as VirusTotal, when available. [51]

Assigning a threat name for a malware that is matched by a signature can be more complicated for linked samples. As previously mentioned, links between samples correspond to cases, where traffic generated by an analysed sample is matched by multiple signatures. This can indicate that the signatures match on different types of traffic of the same malware, which is a desireable finding as it fulfills the first condition of a good signature. On the other hand, the information can indicate that the clustering has produced noisy and inaccurate results. Furthermore, it is possible that the sample submitted for sandbox anlysis is a type of malware that loads a second malware, which generates some network traffic of its own. In order to attribute a threat to linked samples, information that supports the loading of another malware has to be found, or the suspicion can be dismissed. The general procedures for this are manual network traffic analysis on the samples, which attempts to find network traces of download events, and inspection of the sandbox analysis reports.

The main goal of the manual analysis phase is to produce refined and high quality signatures. In the event where a cluster has undergone manual verification for maliciousness and threat composition, the signature is tuned to provide a match as precise as possible. This phase attempts to satisfy the second general condition of a good signature, while keeping the set of matched threats contained to the intended malware. By experimenting with the generated signature constraints and searching the database with different queries, outliers that were possibly discarded in the clustering phase can be discovered and incorporated into the signature. Through this process the signature is refined to both higher accuracy and precision.

At this stage in the analysis process the full potential of the network fingerprint definition can be harnessed through the introduction of server response constraints

into the signatures. The motivation for this is based on the objective of achieving performant detection that is both reliable and accurate. The premise, and optimal condition, for this phase is thus an already established high quality network signature that fulfills the general conditions of a good signature. The benefit of incorporating response constraints to the signatures is the improved reliability through an increased number of constraints. This is amplified by the fact that server responses are delineated by the state machine of the receiving C&C entity and incorporating constraints for a second state machine would thus improve the reliability and resistance of a signature against false positive detection events. Moreover, if server response constraints can be incorporated into a signature, another possibility is to reduce the number of client request constraints in the signature for additional performance gains. Limiting the number of client request constraints in the signature reduces the number of requests that have to be tracked by the IDS in order to perform detection and therefore enables the release of resources earlier. Due to these beneficial properties, the possiblity of including server constraints should be examined and taken advantage of when possible.

4.5 Network fingerprints in Suricata

Signatures that are refined and produced in the manual analysis phase are next deployed to the Suricata IDS. In this phase the signatures are converted into a format that is supported by Suricata and fulfills Suricata's specifications. An important aspect in this phase is the precision of the final converted format, which can have significant effects on the efficiency of applying rules on network traffic. This work won't focus on the actual conversion script, but explains the matching methodology of network fingerprints with Suricata. The conversion takes as its input a set of network signatures, and produces three files that effectively represent the concrete final form of the network fingerprints.

The Suricata IDS is placed on the edge between two networks to perform network traffic monitoring. Traffic is matched by comparing the definitions in Suricata signatures to the monitored traffic. The signatures follow the Suricata signature language format that defines various rules and constraints for when a flow matches, and furthermore specify how a match event is handled. In general, a match event for a packet is triggered when all specified constraints in a signature are fulfilled. Upon starting Suricata, signatures are loaded from a file for which the filepath is defined in the configuration file of the IDS.

The Suricata signature language specification describes multiple keywords that define how a signature should be parsed and applied on the monitored traffic. These keywords can define for example content constraints, perl compatible regular expressions (PCRE) and distance parameters for the matching of subsequent constraints. For example, for HTTP flows these keywords provide an easy way to place contraints on different parts of the HTTP request, such as the "http_uri" keyword for the HTTP URI or "http_method" for the HTTP method. The support for a multitude of different keywords allows the composition of precise signatures and provides a

flexible way of targeting complex structures in packets.

An additional useful feature of Suricata is the chainability of signatures, which allows matching on packet sequences. This chainability is implemented through *flowbits* that are special register values that can be set by a signature for a matching packet, and later verified for by another signature that targets the same flow. Thus by setting a flowbit, a signature that targets a packet at the start of a flow can relay information about the match to another signature that targets a subsequent packet. If the flowbit is not set by the first signature, the second signature is not matched further after verifying the flowbit. By suppressing alerts and relaying match information through flowbits, positive decisions for packet sequences in a flow can be chained until the last Suricata signature for a given flow raises an alert. The additional benefit of chaining signatures with flowbits is that Suricata only has to store a single bit for a sequence of signatures instead of maintaining a complex state in memory for each detector.

The keywords and the chainability of signatures enable the basis for implementing network fingerprints in Suricata. The tuples of a network fingerprint can be chained by setting and checking flowbits in separate Suricata signatures that define and verify the constraints for the values in a tuple. The Suricata signature specification defines the keyword "dsize" that can be used to set a constraint on the payload size of a packet. However, the specification doesn't directly define keywords for the two other required elements in a network fingerprint, i.e. packet index and payload entropy.

In order to match on packet indexes and paylod entropies, the presented system leverages the scriptability of Suricata. Suricata provides an interface for Lua scripting, which is enabled by the just-in-time (JIT) compilation of Lua scripts. Lua scripts can be invoked by Suricata signatures through the use of two Lua specific keywords, "lua" or "luajit", which take as argument the path to a Lua script. The interface exposed to Lua scripts provides visibility on packet payloads of a flow and thereby enables arbitrary calculations to be performed, including the computation of a payload's entropy. The interface implements a "match" function, which is used to set the flowbit of a Suricata signature from Lua. Decisions to stop matching on a flow can therefore be made directly from the Lua script.

Lua scripts have also the capability of exposing values to other Suricata signatures. This can be achieved with the use of *flowint* variables, which store an integer value and make it accessible from other signatures through the keyword "flowint" and the name of the flowint variable as argument. Flowint variables can therefore be used like regular keyword constraints in the matching process. A Suricata signature that is part of a network fingerprint signature chain can thereby verify all necessary tuple value conditions for a given packet in the sequence, i.e. packet index, payload size and entropy. In order to store the entropy values in flowints, a computed entropy value is multiplied by 100 and cut at the decimal point.

The implementation of network fingerprints relies ultimately on three files: two Lua scripts and a signature file. Because Suricata operates on a packet basis and exposes the packet payload to the Lua scripts without a notion of client or server transaction, separate files are required for the computation of client and server entropies and indexes. The Lua files are invoked for each TCP packet from

the signature file by two signatures that are designated for this special purpose. The invocation of the two scripts for client and server transactions exposes the entropies and indexes in the flowints “client.idx”, “server.idx”, “client.entropy” and “server.entropy”. These flowints are available for the matching operation for a packet until the packet is dismissed and new values are computed for the next incoming packet.

An example of a signature file is presented in Appendix C. The signatures in the file are stripped from keywords that are not relevant in the context of this work, but are necessary for Suricata, and the file would therefore not work in a real Suricata environment. In the file, the two Suricata signatures from the top invoke the Lua scripts `fingerprint.lua` and `fingerprint_server.lua`, which are responsible for incrementing the index and computing the payload entropy of an incoming packet. The flowints computed and exposed are used in the four last Suricata signatures that comprise the network fingerprint. The simple form of the network fingerprint contained in the last four Suricata signatures is displayed in Listing 2. Each of the constraints in the listing are included through regular Suricata keywords or through flowints. The sequence in the matching operation is achieved with the flowbits `NF.1` to `NF.3`. When a signature in the sequence matches on a packet, it sets the flowbit that is checked by the next signature in the network fingerprint. In the case of the example, only if the first three signatures have matched on their packets, the flowbit `NF.3` will be set. If the last signature matches its corresponding packet, then it generates alert about a match event.

```
to_server; client.idx=1; dsize=141; entropy=>550;||
to_server; client.idx=2; dsize=97; entropy=>550;||
to_server; client.idx=3; dsize=23<>37;||
to_server; client.idx=4; dsize=65;
```

Listing 2: Simple form of the network fingerprint in Appendix C.

The conversion algorithm available in this phase converts network fingerprints from the format presented in Listing 2 to format presented in Appendix C. In this process the two Lua scripts are created and placed into a shared directory with the generated Suricata signature file. The files are then shipped to the network sensors to be deployed to the Suricata intrusion detection system and to be employed in the detection of encrypted C&C channels.

4.6 Event analysis system

After deployment, the generated network fingerprints undergo a mandatory test phase where the validity of their behaviour is verified. This test phase is enabled by the intrusion detection and malware analysis platform that is available in the context of this research. The context offers a production environment, which consists of multiple real-world customer networks that are comprised of thousands of hosts.

The availability of such framework for testing purposes is atypical to state of the art research projects and thus provides the unique opportunity to perform large-scale testing on the network fingerprint technique. The main goal of the testing phase is to verify that the generated network fingerprints are a feasible detection method for encrypted C&C channels. This is determined by evaluating the effects on the performance of the network sensors, and the qualitive and quantitive analysis on the generated detection events.

Events generated by deployed network fingerprint detectors propagate to an event analysis system that is part of the network analysis system. As mentioned in Section 3.2, this system is used by analysts to inspect generated events and to analyse the behaviour of the detectors. For this purpose, the event analysis system provides a comprehensive overview of a detector and offers detailed information about generated events from all network sensors that employ the detector. This internal analysis system facilitates the manual verification of the behaviour of test mode detectors, and allows a thorough inspection process on the correctness of detectors before they are promoted to a real mode. Promoting a detector to real mode has the notable effect that raised events are further propagated to external systems that are accessible by the customers. Therefore promoting a detector is a significant decision that must be justifyable by the generated events.

The deployment of network signatures to the production environment offers a straightforward testing methodology that provides quantifyable information about the quality of detectors. This information is based on the verification of events that are generated through exposing the detectors to large volumes of user generated network traffic. The verification attempts to determine whether an event generated by a detector matches correctly on the targeted malicious traffic, i.e. it's a true positive detection, or whether the matched traffic is something that was not intended to be targeted, i.e. it's a false positive detection. Based on the assumption that traffic generated by real users is comprised for the most part of benign traffic, this methodology is in principle especially effective in testing detectors for false positive events. The decision whether an event is a true or false positive can be tagged to the event, which is then shown in the combined information for the detector in the event analysis system. A detector typically requires a number of true positive decisions before it can be considered for promotion to real mode.

The verification process for a detector starts when a sensor generates detection events. The starting point for the verification is the network traffic that raised the event. This is enabled by the feature in the sensor, where a buffer of packets that match a network signature are stored and sent as part of the event information to various internal systems. This network traffic is then accessible through the event analysis system along with metadata about the network flow. The decision on the network traffic relies on the knowledge and understanding of the analyst about the detector and the threat that is targeted. In simple cases, where the packet contents in the event data provide strong evidence for a supporting or undermining decision, deeper analysis on the event metadata and detector information may not be necessary, as the knowledge on the malicious network traffic can be sufficient. This can be particularly true for detectors that target C&C communication that

exhibits characteristic features in the traffic. More difficult cases have to take into account additional metadata about the event, such as host and DNS information, IP addresses and port numbers, and related events for the threat or other detectors.

The decision processes for network fingerprint detectors are emphasized by their meticulous type. This is due to the fact that encrypted C&C channels don't exhibit characteristic invariants in the packets payloads and may be difficult to identify reliably based on the network traffic. However, due to this property of the targeted threat type, identification of clear false positive events is simpler: if the event data of a detector that targets encrypted communication contains plaintext strings, the event is likely to be a false positive. Encrypted looking traffic, on the other hand, can be generated next to a malicious threat also by a benign protocol. Verification processes for network fingerprints therefore usually involve deriving a decision from a combined set of available information about the threat, the detector and the events.

The action taken for a detector depends on the decisions that have been produced for the detector. False positive decisions for a detector means that the detector is bad and does not pass the verification process. The action taken in this case is the disabling of the detector, which means that the network sensors stop applying the detector on network traffic. Depending on the false positive events and the number and type of constraints in the network signature, the detector can be discarded completely or alternatively improved through a loop back to the manual analysis phase of the signature generation. When a detector is returned to manual refinement, the new signature has to undergo a new verification phase. However, event information from the bad detector can serve as supporting information for decisions on the events for the improved detector and thereby contribute to the general event verification process.

The action for a detector that has produced true positive decisions depends on the time frame in which the events were generated. A detector, which has been an extended period of time in the test phase and hasn't generated any false positive events, can be considered reliable and thus promoted to real mode. However, if a detector generates true positive events soon after being deployed to the sensors, it proves the accuracy of the detector, i.e. that it matches on malicious traffic, but doesn't guarantee reliably its precision, i.e. that the threat matches only on malicious traffic. Therefore a minimum amount time that a detector has to remain in the test phase should be enforced.

4.7 Summary

Establishing network fingerprint signatures involves a multistep process pipeline, where each stage transforms the input data before passing it on to a subsequent stage. The process is commenced with a protocol filtering step, where the network traffic generated in the sandbox analysis environments is parsed and filtered to fit into the scope of this research. TCP traffic that is not matched by the available protocol matchers is tagged as unknown and transferred to the clustering phase. The clustering phase focuses in the extraction of payload size correlations between client request

sequences. These correlations are described in clusters that are generated with the density-based DBSCAN clustering algorithm. The correlating payload size sequences are combined with payload entropy information in the signature generation stage. This stage is responsible for deriving accurate constraints for the formed clusters and providing simple human-readable signature representations for the clusters. These signatures are visualised in the manual analysis phase, where the signatures undergo a refinement and improvement process. The beneficial properties provided in this phase underline the importance of manual analysis in the generation of network fingerprints, and for network traffic analysis processes in general. The refined network fingerprint signatures are then converted to a format that is supported by the Suricata IDS. The detection in Suricata leverages the scriptability of Suricata through Lua. The converted format is then shipped to the network sensors to be employed in the detection of encrypted C&C traffic. This phase involves a mandatory test period, in which the network fingerprint detectors are verified for reliability and accuracy of detecting the targeted malicious traffic.

5 Evaluation and discussion of the proof of concept

In this chapter the results of the researched proof of concept are presented and discussed. The results are evaluated against the research goals that were introduced in Chapter 3. The first section presents the results of running a number of detectors in the test environment for a test period, and illustrates the performance impact of carrying out Lua supported detection with Suricata. The second section discusses the results and the test environment, and considers the theoretical and practical effects that it induces on the results. The last section provides a short summary of the whole chapter.

5.1 Research results

The main objective of this research was to produce a proof of concept for a network-based detection capability of encrypted C&C channels. This goal was approached by introducing the concept of network fingerprints. The theoretical concept was realized by designing and implementing a multistep analysis pipeline for the generation and verification of network fingerprints. Towards the end of the research five network fingerprints were generated using the built pipeline. These were deployed in test mode to the production environment networks provided by the network detection and malware analysis solution by Lastline Inc. on the 25.01.2016. This section presents two of the detectors and the results that were obtained during the test period.

Detector 1, for which the simple form is shown in Listing 3, targets the Tofsee spambot. The signature consists of four client request packets that all have payload size constraints for unique values, except for the third payload, which targets a range. The two first payloads showed high entropies during the manual analysis, which were included into the signature as threshold values. Tofsee uses a custom encryption algorithm to conceal the cleartext C&C protocol messages. The first client request sent by the bot includes information about the victim machine, such as local time and system version. The subsequent requests can deliver configuration files and various other information extracted by the bot from the victim machine, which can cause variations to the sequence of transmitted payloads. During manual analysis it was noticed, however, that this particular sequence of four requests occurs frequently in the C&C communication generated by Tofsee samples and was therefore targeted in a network fingerprint. [52]

```
to_server; client.idx=1; dsize=141; entropy=>550;||
to_server; client.idx=2; dsize=97; entropy=>550;||
to_server; client.idx=3; dsize=23<>37;||
to_server; client.idx=4; dsize=65;
```

Listing 3: Network fingerprint for the Tofsee detector.

Detector 2 targets check-in activity from the Netwire remote access trojan (RAT) malware. The fingerprint generated against this threat targets the four first requests of the RAT, which are presented in Listing 4. According to the security research article by Da Silva et al., Netwire RAT uses the 256-bit AES algorithm to encrypt its communication. The messages sent by the RAT are comprised of a 4 byte length header, a 1 byte command header and an optional data section. The first request by the RAT is always 69 bytes long and functions as a session initialization and key exchange message with the C&C server. After the key exchange the bot sends a variable length acknowledgement message that is therefore targeted in the signature with the payload size constraint of 57 to 75 bytes. After the second request the malware starts sending periodical heartbeat messages that don't incorporate a data section and are thus only 5 bytes long. These heartbeat messages are targeted with the third and fourth payload constraints in the fingerprint. [53]

```
to_server; client.idx=1; dsize=69; entropy=542<>602;||
to_server; client.idx=2; dsize=57<>75;||
to_server; client.idx=3; dsize=5;||
to_server; client.idx=4; dsize=5;
```

Listing 4: Network fingerprint for the Netwire RAT detector.

The test period spanned 17 days from the deployment date until the 11.02.2016. During this period the network sensors generated altogether 80 events for the five detectors. These events were, however, divided only between the presented Tofsee and the Netwire RAT detectors, where the Tofsee detector generated 71 events and the Netwire RAT detector 9. Therefore the other three detectors are not presented in this section, but are discussed in the next section. The results obtained from the test period are presented in Table 1, where each of the detectors that generated events is displayed on separate rows with the true positive and false positive numbers in the 2 rightmost columns.

Table 1: The number of true positive and false positive detections for each detector.

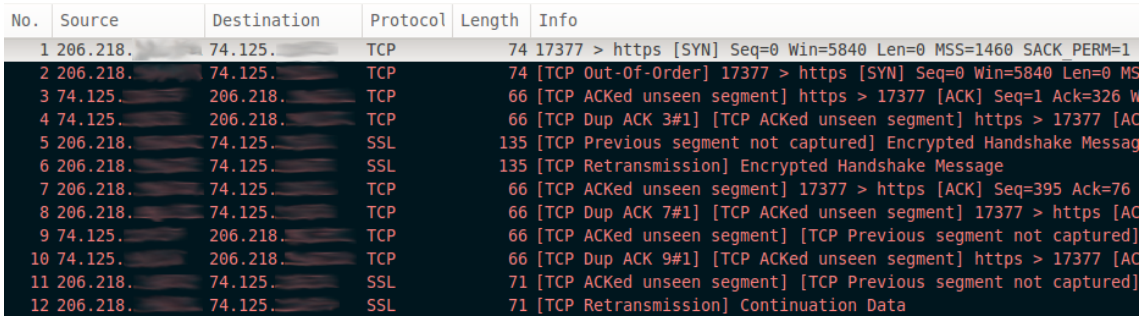
Detector number	# True positives	# False positives
1	71	0
2	0	9

Figure 17: True positive detection for the Tofsee spambot malware.

The results obtained during the test period provide clear indication that the Tofsee detector performs well and should be considered to be promoted to real mode. In the verification process of these events it was confirmed that all 71 of the generated events are true positive detections of the targeted C&C channel. This detector thus proves that the network fingerprint approach for detecting encrypted C&C channels works in practice. Figure 17 illustrates one of the generated events in the event analysis system and displays the network traffic that was detected successfully. It is clear from the figure that the network traffic could not have been detected with conventional pattern-based techniques and would have otherwise passed undetected through the traffic monitor in the IDS if not for the deployed network fingerprint detector.

Detector 2, on the other hand, performed less well in the test period. The Netwire RAT detector generated nine verified false positive events and no true positive events, which are sufficient reasons to not consider promoting the detector. Eight of the nine events were false detections on TLS handshakes and the remaining one matched on a benign HTTP request. The false positive events nevertheless provide useful information about the reliabilty of network fingerprints and how similar false positives could be avoided in the future. Deeper analysis into the nine events revealed that the main probable cause for all false detections were TCP flows that contained retransmissions and out-of-order packet arrivals.

Figure 18 illustrates this problem and displays the packet capture from one of



No.	Source	Destination	Protocol	Length	Info
1	206.218.	74.125.	TCP	74	17377 > https [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1
2	206.218.	74.125.	TCP	74	[TCP Out-Of-Order] 17377 > https [SYN] Seq=0 Win=5840 Len=0 MS
3	74.125.	206.218.	TCP	66	[TCP ACKed unseen segment] https > 17377 [ACK] Seq=1 Ack=326 W
4	74.125.	206.218.	TCP	66	[TCP Dup ACK 3#1] [TCP ACKed unseen segment] https > 17377 [AC
5	206.218.	74.125.	SSL	135	[TCP Previous segment not captured] Encrypted Handshake Messag
6	206.218.	74.125.	SSL	135	[TCP Retransmission] Encrypted Handshake Message
7	206.218.	74.125.	TCP	66	[TCP ACKed unseen segment] 17377 > https [ACK] Seq=395 Ack=76
8	206.218.	74.125.	TCP	66	[TCP Dup ACK 7#1] [TCP ACKed unseen segment] 17377 > https [AC
9	74.125.	206.218.	TCP	66	[TCP ACKed unseen segment] [TCP Previous segment not captured]
10	74.125.	206.218.	TCP	66	[TCP Dup ACK 9#1] [TCP ACKed unseen segment] https > 17377 [AC
11	206.218.	74.125.	SSL	71	[TCP ACKed unseen segment] [TCP Previous segment not captured]
12	206.218.	74.125.	SSL	71	[TCP Retransmission] Continuation Data

Figure 18: TCP retransmissions as probable cause for FP events.

the false posivite events on a TLS handshake. In the figure the payloads that were matched by the signature are transmitted in packets 5, 6, 11 and 12. Looking at the sequence of packets and the information in the capture, it is apparent that packet 6 is a retransmission of packet 5 and similarly packet 12 is a retransmission of packet 11. A reasonable deduction from this is that the false positive events were caused by duplicate packets being sent over the network and the signature targeting similar malicious traffic. Therefore the events were the result of a corner case of benign protocol behaviour, which could not have been taken into account during the signature generation phase due to the limited set of traffic available in the network analysis database. Suricata's flow reassembler might have significance on the false detections as well, but details about the operation of Suricata and finding proof for this hypothesis are outside of the scope of this research and part of future work.

The false positive events of the Netwire RAT detector provide insights about how the false detection events, and potentially similar events in the future, could be avoided. Because eight of the nine events matched on TLS handshakes and one on an HTTP request, a straightforward way to avoid similar false detections is to instruct Suricata to attempt to match on traffic that utilizes ports other than 443 and 80. This approach, however, has a significant downside due to the common evasion technique, where malware connect to well-known ports to bypass firewalls, as was described in Section 2.3. Therefore imposing restrictions on well-known ports can cause false negative detections, i.e. undetected malicious behaviour, and should be based on reliable sources, e.g. reverse engineering a sample. A port constraint for this detector would therefore require analysis on the protocol implementation of the malware to support or dismiss the action.

Another approach to limit the likelihood of false positive events is to increase the number of constraints in the signature. Because the protocol behaviour for this particular threat is known, extending the heartbeat sequence with additional payload size constraints would reduce the probability of matching on further retransmitted packets. This approach, however, is still vulnerable to multiple sequential TCP retransmissions and provides therefore only weak guarantees for improvement. A preferable option would be to instead incorporate server response constraints to the signature and thereby prevent duplicate hits on client TCP retransmissions from triggering a detection. This approach provides an easy solution in the particular case

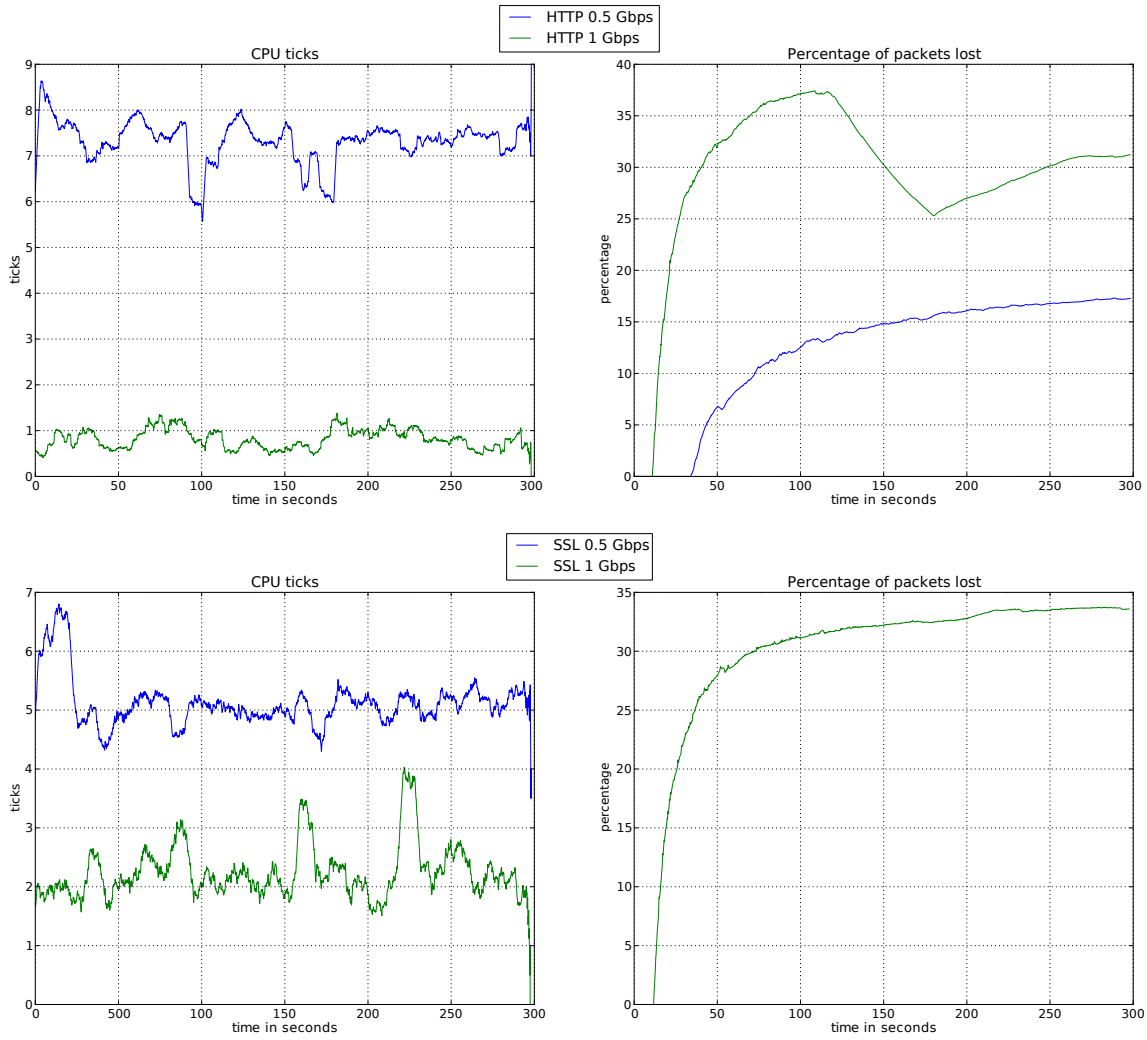


Figure 19: HTTP and SSL performance test with Lua disabled.

of Netwire RAT, because the server responses follow a similar distinctive and easy to match pattern as the client requests.

In addition to qualitative analysis on the events generated by network fingerprint signatures, the effects on the performance of Suricata were also tested. The goal of the performance tests was to measure the effects of carrying out detection in Suricata that leverages Lua scripting. The test setup consisted of a network with two hosts: a client host that generates requests, and a server host that serves resposes to the requests and additionally runs Suricata. The basic test methodology was to run a daemon on the client host that would generate network traffic towards the server and measure the performance of Suricata in processing the requests and the responses in the two test cases where Lua scripts are invoked and when they are not. Each test case was carried out for two different protocols, in this case HTTP and Secure Sockets Layer (SSL), in order to gain further insights about the effects of the protocol. The daemon was instructed to generate and maintain 100 parallel connections towards the server, which in turn was instructed to respond with 100 kilobytes of random

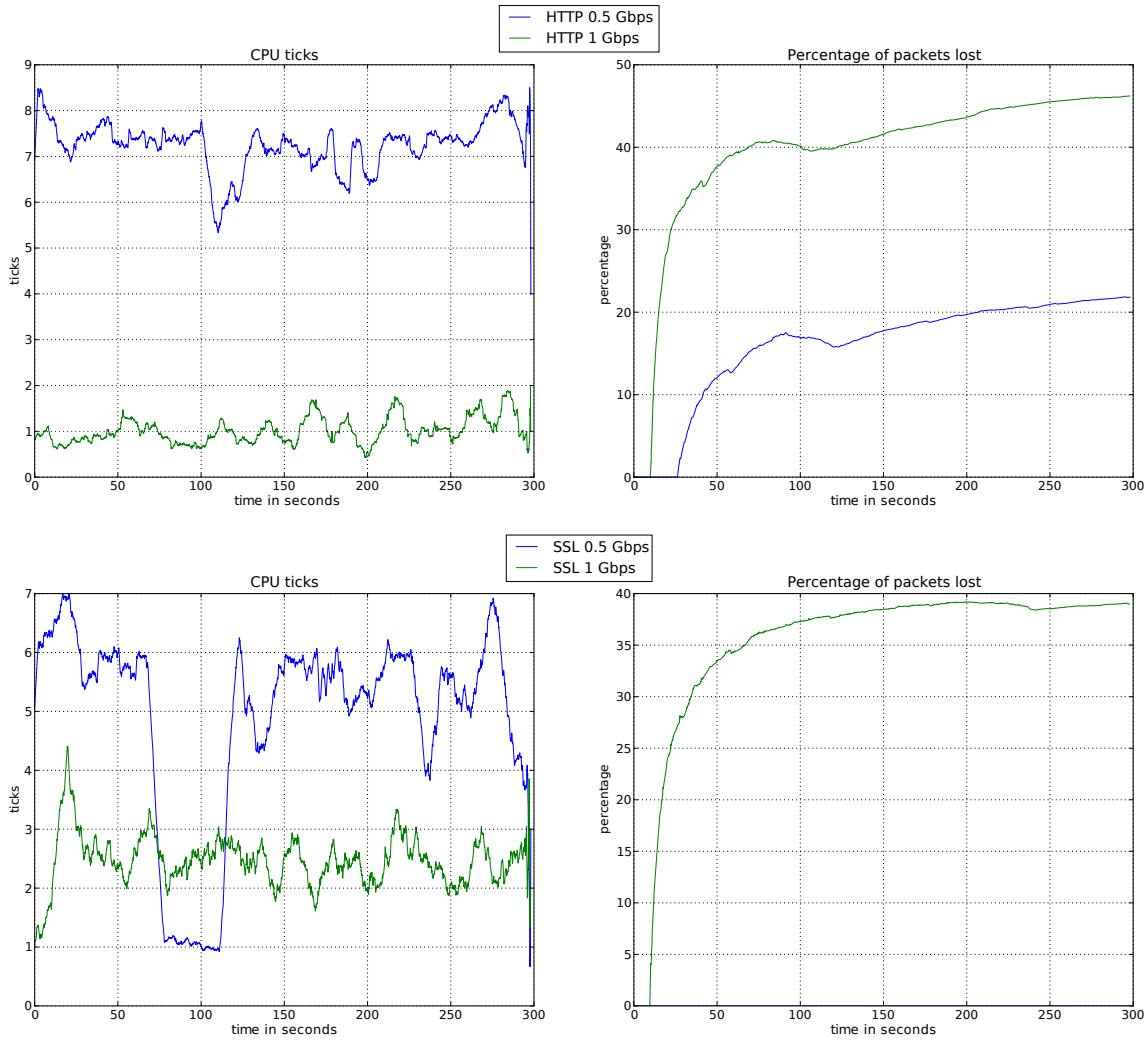


Figure 20: HTTP and SSL performance test with Lua enabled.

gzipped data per transaction. Furthermore each test was run on a 500 Mbps link and a 1 Gbps link to illustrate the behaviour and performance of Suricata in high speed context. Further stress was induced to Suricata by randomising the end point IP addresses and ports for each flow so that different optimising techniques could not be potentially utilized.

The two performance test scenarios are depicted in Figures 19 and 20. The top row in each figure depicts the test measurements with the HTTP protocol and the bottom row with SSL. The left column in the figures portrays the CPU load in absolute number of ticks, while the right column shows the cumulative packet loss percent. In each graph the 500 Mbps test is shown in green and the 1 Gbps test in blue.

The top row in Figure 19 for the 500 Mbps speed shows the development of the cumulative packet loss. During the 300 second test period the amount of packet lost grows steadily and reaches approximately 17 % at the end of the test. This observation tells that Suricata starts losing packets after a period when the load

is constant. This is because Suricata reaches its the maximum buffering capacity, which occurs for the 500 Mbps test a short time later than in the 1 Gbps test. For the 1 Gbps test the packet loss rises during the first 100 seconds to approximately 37 %, after which the test for unknown reasons "recovers" causing Suricata to be able to process more packets. After a short period, however, the cumulative packet loss starts growing again and reaches approximately 32 % at the 300 second mark. The amount of packet loss is reflected in the CPU load graphs. At lower speed Suricata is able to receive incoming packets in higher quantity and thus the stress on the CPU during the packet processing is higher. At the 1 Gbps speed Suricata drops more packets, which then don't end up being processed, and the CPU load is therefore lower. Similar observations are true for SSL, although the packet loss for the 500 Mbps link coulnd't be measured. The packet loss of the 1 Gbps test shows that approximately 35 % of the received packets are dropped.

Figure 20 shows the performance measurements for the test case where Lua was enabled. For the 500 Mbps test with HTTP the packet loss rises to about 22 %, while in the 1 Gbps test the packet loss reaches approximately 46 %. In the 1 Gbps SSL test the packet loss grows up to 37 %. Comparing the two test cases shows that enabling Lua causes a 5–9 % increase in packet loss for HTTP and a 2 % loss for SSL. The tick count, which describes the CPU load during packet processing, remains approximately at the same levels for all tests with HTTP. Enabling Lua computations on a sensor therefore doesn't affect the load on the CPU after packets have been accepted. For SSL, on the other hand, the average CPU load experiences a slight increase.

5.2 Discussion

The test environment available for the research provided the unique and significant opportunity to test the established network fingerprint technique in real-world networks. The benefit of this kind of environment is invaluable for the qualitative testing process as it exposes the tested signatures to large volumes of benign traffic. User generated real-world network traffic includes edge and corner cases in the communication that would normally be impossible to test for in a separated test environment. Thus, the environment used for the testing of the proof of concept in this research is an invaluable asset that provides supporting arguments for the correct functionality of the presented proof of concept system.

The test environment facilitated especially the testing for false positive events. This feature is desireable, as deriving a theoretical model for the combination of the number and type of signature constraints for a reliable signature is a difficult task, as mentioned in Section 3.2. Based on the conducted experiments and the obtained results that were presented in the previous section, it can be concluded that at least four client transaction payload size constraints and two entropy contraints can be enough to produce an accurate and well performing detector. The specifics of these constraints and their combination, however, have an effect on the reliability of the detector, as was observed for the Netwire RAT detector. False positive detections

for test detectors nevertheless provide advantageous insights into the context traffic that generated the events, and thereby enable the improvement of the detectors and the signature generation methodologies.

An important aspect to take into account, when making inferences about the minimum requirements of constraints for a reliable detector, is that the available test environment can not be used to test for a given threat at will. The infections in a network are a variable in the test environment that is uncontrollable and unpredictable. Therefore the correctness of a signature in detecting certain malware communication can not be planned to be concluded through the number of generated true positive events in a certain test period. Instead, detectors may be left in the testing phase for extended periods of time until supporting events for a promoting or disabling action are received. A lack of events can thus indicate either that (i) the detector is faulty and therefore doesn't match benign or malicious behaviour; or that (ii) the detector is not very bad, as it doesn't generate false positive events, but the production environment networks don't contain active infections of the targeted malware.

Malware activity is often limited to short periods of time until the infections have been cleaned, and therefore the C&C activity targeted with a detector that would serve as the grounds for a decision is not available anymore. Knowledge of this possibility in the test environment enables the undesireable situation where faulty detectors are assumed to be functioning properly, but simply don't receive events due to the lack of malware activity. Such assumptions can lead to a scenario where a malware is still active in the test environment, but is not matched due to the faultiness of the detector. These events are false negative detections and are impossible to measure in the context of the available test environment. These indications, or a combination of both, may also be the decisive factor for why the three other detectors, which had similar constraints as the two other ones, didn't produce any events and therefore can't be reasoned on more extensively. A general deduction from this is that any event for a detector, i.e. true positive or false positive, is more useful and therefore more favorable than not receiving events at all.

The performance tests show that the network fingerprint capability induces a small performance penalty. The performance penalty was most visible in the cumulative packet loss, which experienced an increase of 2–9 % for both tested protocols. The CPU load on the other hand remained unaffected in the HTTP test with Lua and showed only a slight increase for SSL. These penalties to the performance of Suricata form only a minor drawback for incorporating the capability to detect encrypted C&C channels. Due to their evasive characteristics and difficulty of matching with conventional pattern-based techniques, the threats targeted with network fingerprints may otherwise go completely undetected. The tests carried out in this research alone showed that 71 C&C channels would not have been detected if the technique wasn't employed. A general conclusion from this consideration is that the benefit of employing network fingerprints outweighs the small performance penalties caused by it.

The relatively limited time frame that was available for the experimental phase had unfortunate effects on the testing. A major feature that couldn't be incorporated

to the testing phase due to its late introduction to the analysis system was the support for server transaction constraints. As has been previously reasoned in this work, server transaction constraints in network fingerprints theoretically provide significant additional reliability and robustness against false positives. Further tests that measure the improvements brought by server transaction constraints and provide a comparison to client transaction-only signatures are left for future work.

Another unfortunate effect of the short test period was that a large scale testing with a greater number of detectors could not be carried out. The main drawback of this is the lost opportunity to collect additional quantitative information about network fingerprint detectors. Although the obtained results are toned down by the small scale of the test phase, the carried out performance measurements and the generated events allow drawing the following conclusions:

1. Network fingerprints are a performant technique that is feasible to deploy to large production environment networks

2. Network fingerprints are capable of accurately detecting real malicious encrypted C&C channels

Even though the approach presented in this work has proven to be an effective and performant way of detecting customly encrypted C&C channels, there exist some evasions that can be employed by attackers to thwart network fingerprint-based detection. As network fingerprints ultimately only depend on extracted message length and entropy information in C&C communication, in order to preserve confidentiality of the transmitted C&C messages, useful attacks against the detection mechanism can only focus on tampering message lengths. The general aim of such attacks would be to confuse the clustering algorithm by reducing the correlation between extracted feature vectors, which would lead to a lower number of identified clusters and ultimately a lower number of generated signatures.

Reducing the correlation between extracted feature vectors can be approached by introducing randomness to the C&C message sequences. This could be achieved for example by appending or prepending padding of random length to each message, and including the length and position of the padding with the message. This technique has been previously employed by Zeus P2P [12] and Poison Ivy [54] to thwart traffic analysis. Another option is to extend the C&C protocol of the malware to inject additional redundant random length messages at arbitrary positions in the message sequence. These messages can be simply discarded by the communicating endpoints after decryption, but the defender wouldn't be able distinguish them from other C&C messages. To the knowledge of the author this technique hasn't been employed by any malware thus far.

6 Conclusions

The threat landscape of the Internet has evolved drastically in the past two decades. The number of malware has grown to hundreds of millions unique samples in 2016 and new threats are introduced continuously by malicious actors. The evolution of the threat landscape has been induced by technological advancements that have resulted in the proliferation of personal computers and other connected digital devices. This proliferation has opened a new lucrative business opportunity for criminals and as a result caused a shift in the type of malicious actors in the Internet. Malware are increasingly developed by cybercriminal groups that mirror legitimate businesses in their organisational structure and processes. These groups target large masses of users with advanced techniques in the effort to reach maximal returns of investment.

Malware infections that leverage remote controllability through Command and Control communication have become the primary means for financially motivated cybercriminals to carry out their operations. Through large-scale infection campaigns and remote controllability victim machines can be harnessed to perform profitable malicious activities, such as theft of confidential information, spam email distribution or performing disruptive attacks against networks. The success of these operations relies on the availability of a robust and reliable C&C infrastructure that allows malicious entities to issue commands to the bot-infected machines. Protecting this infrastructure against disruptions and take downs is therefore an integral requirement for cybercriminals in order to ensure continuity of their operations.

Malware C&C communication has been the target of extensive study and defense efforts by security professionals and researchers. Due to the difficulty of preventing initial malware infections, detecting and identifying post-compromise C&C activity offers an effective way of preventing financial damage and theft of confidential data. Driven by the defense efforts by the security industry, malware authors have been forced to evolve and come up with novel ways to evade detection. This has shaped into a constant battle between the attackers and the defenders, where the attackers in response to defenders' endeavours repeatedly produce innovative and more resilient techniques to thwart detection.

Conventional network-based C&C detection techniques rely on invariants in the observed network traffic. Application-layer carrier protocols for C&C communication, such as HTTP and IRC, commonly exhibit tags and delimiters into the observeable traffic, which can be targeted with pattern-based identification techniques. However, developments in malware suggest that C&C communication is increasingly characterized by properties that provide covertness for the C&C protocol and attempt to hide invariants from the traffic. A commonly utilized implementation of this methodology is the use of a non-descriptive carrier protocol to carry the C&C messages, which have been obfuscated with a custom encryption algorithm. This new approach for performing covert C&C communication effectively thwarts conventional pattern-based detection techniques and imposes a detection gap to current intrusion detection systems.

Inspired by the new evasion mechanism, this work presents a novel technique for the detection of encrypted C&C channels. The presented technique, network fingerprints, leverages side-channel information leaked in transmitted C&C messages and uses these to identify similar sequences of C&C protocol messages in networks. This work focuses in the creation of a proof of concept system that is able to analyse network traffic and to automatically produce signature candidates for network fingerprints. This process involves the designing and constructing of a multistep analysis pipeline, which processes raw traffic captures and produces network fingerprint signatures for the Suricata intrusion detection system.

This methodology for C&C detection was put into practice and a proof of concept system was realized. The established analysis pipeline incorporates a protocol filtering stage at the beginning, where source data provided by sandbox analysis environments is filtered in order to produce a dataset that fits the scope of the work. The filtering is performed with protocol parsers that discard known application-layer protocol implementations, and thereby produce a dataset that consists of network traffic that implements custom or unknown application-layer protocols. This data is input to a clustering phase where payload size sequences are extracted from traffic flows and used as the correlating factor in a grouping operation. The clustering phase produces sets of grouped network flows, where each group is characterized by an intercorrelating sequence of payload sizes. This information is utilized in a signature generation phase, where additional side-channel information, particularly entropy, is extracted from the grouped flows and used as additional properties to provide an accurate description of the groups. The signature generation phase produces network signatures that incorporate the payload size and entropy information of the clustered packet sequences. The automatically generated network signatures undergo manual analysis where they are refined and improved to match more accurately on respective targeted threats. This process is facilitated by a specially built visualisation of the clusters that allows a comprehensive and interactive analysis on the created signatures. After manual analysis, the refined signatures are converted into the signature format supported and employed by the Suricata intrusion detection system. This process creates an efficient representation of the network fingerprints, which leverages the scriptability of Suricata with Lua. The converted signatures are then distributed and deployed to network sensors that employ the generated network fingerprints in network intrusion detection.

The proof of concept system was tested and validated using the production environment that was available in the context of this work. Several network fingerprint signatures were deployed to Suricata sensors in real-world networks and validated against the expectated outcomes that were set for the technique. During the testing period altogether 80 events were generated by two of the five network fingerprints that were deployed. One detector produced 71 true positive detection events, while the other produced 9 false positive events. Manual verification of the true positive events confirmed that the traffic matched with the detector would evade conventional pattern-based detection. The 9 false positive events indicate that the requirements for reliable and accurate network fingerprints still need refining and serve as a valuable asset for future development. The effects on the performance of Suricata were verified

through automatic performance tests. In the tests Suricata was subjected to heavy loads of HTTP and SSL traffic when running a configuration that supports network fingerprints and without. The test results show that the performance penalties induced by the network fingerprints on Suricata include of a 2–9 % overall increase in packet loss and a minor increase in the CPU load. Even though the proof of concept could be tested only for a short period of time and with a very limited set of detectors, the results indicate that the established network fingerprint technique fulfills the goals set for this research and constitutes a feasible approach for detecting encrypted C&C channels.

References

[1] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your Botnet is My Botnet: Analysis of a Botnet Takeover," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 635–647. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653738

[2] Fortinet, "Cybercrime Report," Available: http://www.fortinet.com/sites/default/files/whitepapers/Cybercrime_Report.pdf, 2013, Retrieved Jan 3, 2016.

[3] J. Franklin, "An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 375–388. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315292

[4] K. Rieck, G. Schwenk, T. Limmer, T. Holz, and P. Laskov, "Botzilla Detecting the "Phoning Home" of Malicious Software," New York, NY, USA, pp. 1978–1984, 2010. [Online]. Available: http://doi.acm.org/10.1145/1774088.1774506

[5] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, "Spamming Botnets: Signatures and Characteristics," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 171–182. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402979

[6] F. C. Freiling, T. Holz, and G. Wicherski, "Botnet Tracking: Exploring a Root-cause Methodology to Prevent Distributed Denial-of-service Attacks," in *Proceedings of the 10th European Conference on Research in Computer Security*, ser. ESORICS'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 319–335. [Online]. Available: http://dx.doi.org/10.1007/11555827_19

[7] F-Secure Labs, "Blackenergy & Quedagh: The convergence of crimeware and APT attacks," Available: https://www.f-secure.com/documents/996508/1030745/blackenergy_whitepaper.pdf, 2014, Retrieved Jan 3, 2016.

[8] Fortinet, "Anatomy of a Botnet," Available: http://www.fortinet.com/sites/default/files/whitepapers/Anatomy-of-a-Botnet-WP.pdf, 2013, Retrieved Dec 16, 2015.

[9] QinetiQ, "Command & Control: Understanding, denying, detecting," Available: http://www.cpni.gov.uk/documents/publications/2014/2014-04-11-cc_qinetiq_report.pdf, November 2014, Retrieved Jan 12, 2016.

[10] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[11] N. Falliere and E. Chien, "Zeus: King of the Bots," Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/zeus_king_of_bots.pdf, 2009, Retrieved Jan 11, 2016.

[12] C. Rossow and C. J. Dietrich, "ProVeX: Detecting Botnets with Encrypted Command and Control Channels," in *Proceedings of the 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 21–40. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39235-1_2

[13] "Suricata - Features," Online: http://suricata-ids.org/features/, Last accessed Jan 12, 2016.

[14] J. Gardiner, M. Cova, and S. Nagaraja, "Command & Control: Understanding, Denying and Detecting," *CoRR*, vol. abs/1408.1136, 2014. [Online]. Available: http://arxiv.org/abs/1408.1136

[15] N. Hachem, Y. Ben Mustapha, G. Granadillo, and H. Debar, "Botnets: Lifecycle and Taxonomy," in *Network and Information Systems Security (SAR-SSI)*, May 2011, pp. 1–8.

[16] Microsoft, "Security Intelligence Report: The evolution of malware and the threat landscape – a 10-year review," Available: https://www.microsoft.com/security/sir/story/#!10year, February 2012, Retrieved Jan 3, 2016.

[17] AV-TEST, "Statistics - Malware," Online: https://www.av-test.org/en/statistics/malware/, Last accessed Jan 3, 2016.

[18] Symantec, "ISTR20 – Internet Security Threat Report," Available: http://www.fortinet.com/sites/default/files/whitepapers/Cybercrime_Report.pdf, 2015, Retrieved Jan 11, 2016.

[19] R. Perdisci, W. Lee, and N. Feamster, "Behavioral Clustering of HTTP-based Malware and Signature Generation Using Malicious Network Traces," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 26–26. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855737

[20] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring Internet Denial-of-service Activity," *ACM Trans. Comput. Syst.*, vol. 24, no. 2, pp. 115–139, May 2006. [Online]. Available: http://doi.acm.org/10.1145/1132026.1132027

[21] F-Secure Labs Threat Intelligence, "The Dukes: 7 years of Russian cyberespionage," 2011, Retrieved Jan 3, 2016.

[22] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A Multifaceted Approach to Understanding the Botnet Phenomenon," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 41–52. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177086

[23] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang, "On the analysis of the Zeus botnet crimeware toolkit," in *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*, August 2010, pp. 31–38.

[24] J. Wyke, "What is Zeus?" Available: https://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/Sophos%20what%20is%20zeus%20tp.pdf?la=en.pdf?dl=true, May 2011, Technical Report, SophosLabs. Retrieved Jan 20, 2016.

[25] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos, "Highly resilient peer-to-peer botnets are here: An analysis of Gameover Zeus," in *Malicious and Unwanted Software: "The Americas" (MALWARE), 2013 8th International Conference on*, October 2013, pp. 116–123.

[26] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M. Z. Rafique, M. A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, and G. M. Voelker, "Manufacturing Compromise: The Emergence of Exploit-as-a-service," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 821–832. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382283

[27] N. Daswani and M. Stoppelman, "The Anatomy of Clickbot.A," in *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, ser. HotBots'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 11–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=1323128.1323139

[28] D. Dagon, G. Gu, C. Lee, and W. Lee, "A Taxonomy of Botnet Structures," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, December 2007, pp. 325–339.

[29] K. S. Killourhy, R. A. Maxion, and K. M. C. Tan, "A Defense-Centric Taxonomy Based on Attack Manifestations," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, ser. DSN '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 102–. [Online]. Available: http://dl.acm.org/citation.cfm?id=1009382.1009726

[30] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. Bellevue, WA: USENIX,

2012, pp. 491–506. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/antonakakis

[31] D. Dittrich and S. Dietrich, "Command and Control Structures in Malware: From Handler/Agent to P2P," *;login;, the Magazine of USENIX*, vol. 32, Number 6, 2007. [Online]. Available: http://c59951.r51.cf2.rackcdn.com/5553-525-dittrich.pdf

[32] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets: Overview and case study," in *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, ser. HotBots'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1323128.1323129

[33] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, "SoK: P2PWNED - Modeling and Evaluating the Resilience of Peer-to-Peer Botnets," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, ser. SP '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 97–111. [Online]. Available: http://dx.doi.org/10.1109/SP.2013.17

[34] P. Mockapetris, "Domain Names - Concepts and Facilities," RFC 1034, November 1987.

[35] J. Nazario and T. Holz, "As the net churns: Fast-flux botnet observations," in *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, October 2008, pp. 24–31.

[36] D. Schwarz, "Bedep's DGA: Trading Foreign Exchange for Malware Domains," [Online]. Available https://asert.arbornetworks.com/bedeps-dga-trading-foreign-exchange-for-malware-domains/, Last accessed Jan 22, 2016.

[37] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting Malware Infection Through IDS-driven Dialog Correlation," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 12:1–12:16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1362903.1362915

[38] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-independent Botnet Detection," in *Proceedings of the 17th Conference on Security Symposium*, ser. SS'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 139–154. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496711.1496721

[39] C. J. Dietrich, C. Rossow, and N. Pohlmann, "CoCoSpot: Clustering and Recognizing Botnet Command and Control Channels Using Traffic Analysis," *Computer Networks*, vol. 57, no. 2, pp. 475–486, February 2013. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2012.06.019

[40] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting Botnet Command and Control Servers Through Large-scale NetFlow Analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 129–138. [Online]. Available: http://doi.acm.org/10.1145/2420950.2420969

[41] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 191–206. [Online]. Available: http://dx.doi.org/10.1109/SP.2010.20

[42] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering, "IPv6 Flow Label Specification," RFC 3697, March 2004.

[43] J. Goubault-Larrecq and J. Olivain, "Detecting Subverted Cryptographic Protocols by Entropy Checking," 2006.

[44] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic Analysis of Malicious Code," *Journal in Computer Virology*, vol. 2, no. 1, pp. 67–77, 2006. [Online]. Available: http://dx.doi.org/10.1007/s11416-006-0012-2

[45] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. van Steen, F. C. Freiling, and N. Pohlmann, "Sandnet: Network Traffic Analysis of Malicious Software," in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, ser. BADGERS '11. New York, NY, USA: ACM, 2011, pp. 78–88. [Online]. Available: http://doi.acm.org/10.1145/1978672.1978682

[46] P. Ramos, "Dorkbot: Hunting Zombies in Latin America," Available: http://go.eset.com/us/resources/white-papers/Ramos-VB2012.pdf, 2012, Retrieved Feb 5, 2016.

[47] H. Xu, "DorkBot, a Twin Botnet of NgrBot," Online: https://blog.fortinet.com/post/dorkbot-a-twin-botnet-of-ngrbot, Last accessed Feb 5, 2016.

[48] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." AAAI Press, 1996, pp. 226–231.

[49] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[50] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated Classification and Analysis of Internet Malware," in *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*, ser. RAID'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 178–197. [Online]. Available: http://dl.acm.org/citation.cfm?id=1776434.1776449

[51] Kruegel, C., "How To Build An Effective Malware Analysis Sandbox," Online: http://labs.lastline.com/different-sandboxing-techniques-to-detect-advanced-malware, Last accessed Feb 10, 2016.

[52] Mi, R., "Tofsee botnet," Online: https://www.virusbulletin.com/virusbulletin/2014/04/tofsee-botnet, Last accessed Feb 10, 2016.

[53] Da Silva, P. and Downs, R. and Olson, R., "New Release: Decrypting NetWire C2 Traffic," Online: http://researchcenter.paloaltonetworks.com/2014/08/new-release-decrypting-netwire-c2-traffic/, Last accessed Feb 10, 2016.

[54] M. Yates, M. Scott, B. Levene, and J. Miller-Osborn, "New Poison Ivy RAT Variant Targets Hong Kong Pro-Democracy Activists," Online: http://researchcenter.paloaltonetworks.com/2016/04/unit42-new-poison-ivy-rat-variant-targets-hong-kong-pro-democracy-activists/, Last accessed Sep 24, 2016.

A DBSCAN implementation

```
#!/usr/bin/env python
import numpy
from sklearn.neighbors import KDTree


class DBSCAN(object):
    UNVISITED = 0
    NOISE = -1

    def __init__(self, eps=5, minpts=5):
        """
        :param eps: radius that defines the neighborhood
            of a point
        :param minpts: minimum number of points in a
            neighborhood to create a cluster
        """
        self.eps = eps
        self.minpts = minpts

        self.neighborhoods = None
        self.labels = None
        self.core_points = None
        self.tree = None

    def scan(self, dataset):
        """
        Run dbscan on a given dataset. Returns a list of
        labels, where each label at an index i points the
        cluster number for the point at index i in the
        dataset. The algorithm uses euclidean distance as
        the distance metric for the region query.

        :param dataset: numpy.ndarray, set of points in
            space. Rows correspond to points,
            while columns specify the offset
            on the axes.
        :rtype: numpy.array, numpy.array, integer
        :returns: A list of labels (1-N for clusters, -1
            for outliers), list of indexes of the
            cluster corepoints, number of clusters
        """
        self.tree = KDTree(dataset)

        self.labels = numpy.zeros(
```

```
45              dataset.shape[0], dtype="int64")
46          self.core_points = numpy.zeros_like(
47              self.labels, dtype="bool")
48          nclusters = 1
49
50          for index, dummy in enumerate(dataset):
51              if self.labels[index] != self.UNVISITED:
52                  continue
53
54              is_cluster = self._expand_cluster(
55                  dataset, index, nclusters)
56              if is_cluster:
57                  nclusters += 1
58
59          return self.labels, self.core_points, nclusters-1
60
61      def _get_neighbors(self, dataset, index, nclusters):
62          """
63          Find unvisited neighbors, or neighbors that have
64          been tagged as noise. Don't return points that
65          already belong to other clusters to avoid
66          problems with border points.
67
68          :param dataset: numpy.ndarray, set of points in
69              space. Rows correspond to points,
70              while columns specify the offset
71              on the axes.
72          :param index: Offset in the dataset for the
73              current point.
74          :param nclusters: Current cluster number.
75          :rtype: list
76          :returns: list containing indexes of all
77              neighboring points that are within a
78              self.eps radius of the point
79          """
80          neighbors = self.tree.query_radius(
81              dataset[index], self.eps)[0].tolist()
82
83          return [
84              i for i in neighbors
85              if self.labels[i] in [
86                  nclusters, self.UNVISITED, self.NOISE]]
87
88      def _expand_cluster(self, dataset, index, nclusters):
89          """
90          Actual clustering function. Decide whether a point
91          forms a cluster by searching its neighbors and
92          their neighbors. Mark all such neighbors
```

```
93              belonging to the same cluster and points that have
94              more than self.minpts neighbors as core points of
95              the clusters.
96
97              :param dataset: numpy.ndarray, set of points in
98                  space. Rows correspond to points,
99                  while columns specify the offset
100                 on the axes.
101             :param index: Offset in the dataset for the
102                 current point.
103             :param nclusters: Current cluster number.
104             :rtype: Boolean
105             :returns: True if the current point forms a
106                 cluster, else False.
107             """
108             neighbors = self._get_neighbors(
109                 dataset, index, nclusters)
110
111             if len(neighbors) >= self.minpts:
112                 # Hunt down identicals in order not to choke
113                 # on them in the loop
114                 identicals = self._get_identical_points(
115                     dataset, index, neighbors)
116
117                 # Identicals form the base of the new cluster
118                 processed = set(identicals)
119                 self.labels[identicals] = nclusters
120                 self.core_points[identicals] = True
121
122                 add = processed.add
123                 for neighbor in neighbors:
124
125                     if neighbor not in processed:
126                         add(neighbor)
127
128                         self.labels[neighbor] = nclusters
129
130                         extended = self._get_neighbors(
131                             dataset, neighbor, nclusters)
132
133                         if len(extended) < self.minpts:
134                             continue
135
136                         identicals = \
137                             self._get_identical_points(
138                                 dataset, neighbor, extended)
139
140                         processed = \
```

```
141                             processed.union(identicals)
142
143                         self.labels[identicals] = nclusters
144                         self.core_points[identicals] = True
145
146                         neighbors += (list(set(extended) -
147                                            set(neighbors)))
148
149                 return True
150             else:
151                 self.labels[index] = self.NOISE
152                 return False
153
154         def _get_identical_points(self, dataset, index,
155                                   neighbors):
156             """
157             Get identical points to a point. The returned list
158             always contains also the point itself.
159
160             :param dataset: numpy.ndarray, set of points in
161                 space. Rows correspond to points,
162                 while columns specify the offset
163                 on the axes.
164             :param index: Offset in the dataset for the
165                 current point.
166             :param neighbors: List of point indexes in the
167                 dataset.
168             :rtype: list
169             :returns: List containing identical elements with
170                 a point, i.e. points that share same
171                 coordinates as the point.
172             """
173             point = dataset[index]
174             return [
175                 i for i in neighbors
176                 if numpy.array_equal(dataset[i], point)]
```

B Pseudocode for computation of signature tuples

```
ENTROPY_MARGIN = 0.2
ENTROPY_THRESHOLD = 6.6
STDEVIATION_THRESHOLD = 0.1
tuples = []

for i in range(1, D):
    tuple_values = []

    # Insert payload size constraint
    if len(set(payload_sizes[i])) == 1:
        # Unique value
        tuple_values.append(str(payload_sizes[i][0]))
    else:
        # Range
        min_ = min(payload_sizes[i])
        max_ = max(payload_sizes[i])
        range_ = "{:d}<>{:d}".format(min_ - 1, max_ + 1)
        tuple_values.append(range_)

    # Insert entropy constraint
    if len(set(entropy_values[i])) == 1:
        # Unique value
        entropy = entropy_values[i][0]
        range_ = "{:d}<>{:d}".format(
                entropy - ENTROPY_MARGIN,
                entropy + ENTROPY_MARGIN)
        tuple_values.append(range_)
    elif stdeviation <= STDEVIATION_THRESHOLD:
        # Small standard deviation
        if mean(entropy_values[i]) >= ENTROPY_THRESHOLD:
            min_ = min(entropy_values[i])
            range_ = ">{:d}".format(min_ - stdeviation)
            tuple_values.append(range_)
        else:
            min_ = min(entropy_values[i])
            max_ = max(entropy_values[i])
            range_ = "{:d}<>{:d}".format(
                    min_ - stdeviation,
                    max_ + stdeviation)
            tuple_values.append(range_)

    tuples.append(tuple(tuple_values))
```

C Suricata signatures for network fingerprints

```
# Network fingerprint client transaction Lua script
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Network
   fingerprint client transaction Lua script";
   flow:established, to_server; flowbits:set,fingerprint_lua;
   flowint:fingerprint_disable,notset; lua:fingerprint.lua;
   flowbits:noalert;)

# Network fingerprint server transaction Lua script
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"Network
   fingerprint server transaction Lua script";
   flow:established, to_server;
   flowbits:set,fingerprint_server_lua;
   flowint:fingerprint_disable,notset;
   lua:fingerprint_server.lua; flowbits:noalert;)

# Network fingerprint Suricata signatures
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Network
   fingerprint signature #1 (no alert)";
   flowbits:isset,fingerprint_lua;
   flow:established,to_server; flowint:client.idx,==,1;
   dsize:141; flowint:client.entropy,>,550;
   flowbits:set,NF.1; flowbits: noalert;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Network
   fingerprint signature #2 (no alert)"; flowbits:isset,NF.1;
   flowbits:isset,fingerprint_lua;
   flow:established,to_server; flowint:client.idx,==,2;
   dsize:97; flowint:client.entropy,>,550;
   flowbits:unset,NF.1; flowbits:set,NF.2; flowbits: noalert;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Network
   fingerprint signature #3 (no alert)"; flowbits:isset,NF.2;
   flowbits:isset,fingerprint_lua;
   flow:established,to_server; flowint:client.idx,==,3;
   dsize:23<>37; flowbits:unset,NF.2; flowbits:set,NF.3;
   flowbits: noalert;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Network
   fingerprint signature #4"; flowbits:isset,NF.3;
   flowbits:isset,fingerprint_lua;
   flow:established,to_server; flowint:client.idx,==,4;
   dsize:65; flowbits:unset,NF.3;)
```